# LanBox

# Reference Manual

# version 2.0a1

**About Engines (layers) ?**

**Initialize engines**
A LanBox has multiple identical layers, they only differ in priority. For now we choose the top layer (the highest priority). First, this engine need to be initialized for data output. Note: In the LanBox-LCX and LCM a new layer is always initialized to the default values.

The engine's status can be set. This status has five elements (Outputting, Sequencing, Fading, Soloing and locked). Elements Outputting must be on, sequencing should probably be on (unless you want a edit layer), fading should probably be on (unless you do not want to use fades), Soloing should probably be off (for more information on the solo status see the chapter on soloing) and locked is default off except on a LC. Modifying the engine's status can be done with the **EngineSetOutput**, **EngineSetSequencing**, **EngineSetFading** and **EngineSetSolo** command.

If you decide to use fades, you need to do three things; Enable the engine for fading with the **EngineSetFading** (see also above) command, Set the fade type with the **EngineSetFadeType** command and the fade time with the **EngineSetFadeTime** command.

The engine's mix mode must be set. The mix mode determines how this engine's data is mixed with the other engines data. There are four mix modes (Copy, Mix-up, Mix-down, Transparent). If you want to be sure you'll see the output, use the Copy mode. Setting the Mix mode can be done with the **EngineSetMixMode** command.

**Initialize channel(s)**
If Auto Output is not on, you have to enable one or more light channels outputs with commands. In order to use light channels in an engine, they must be enabled before it can be used. Therefore each channel has an Output enabled attribute. Each channel that is used should be enabled. As long as the channel is enabled the engine will process the channel. The Output attribute can be set with the **ChannelSetOutputEnable** command.

The engine now ready to handle the channel data. This whole initializing procedure needs to be done only once, unless you want to change parameters.

**8 or 16 bit address mode**
While the classic LanBox-LC could only address up to 250 light channels, the new LCM and LCX boxes goes beyond that. This means that if you want to address above 255, a channel should be a 16 bit value (High byte first). For LanBox-LCX and LCM there is a command **Common16BitMode** (65 hex ) in order to set the box so it will accept 16 bit channel addresses.

**Setting a DMX value on the output of a DMX channel.**
Although this may seem like a simple action, a lot of things have to happen before we can accomplish this. We will assume here that nothing has been initialized or has been set to a defined state, so we will do a lot of thing that will normally not be necessary or need to be done only once.

First we need to know the path the data has to travel so we can activate all the components along the way.

**Initialize engine**
The LanBox (LCX, LCE, LCM) has 8 (31) identical engines, they only differ in priority. For now we choose engine A (default the highest priority). First, engine A need to be initialized for data output (default already don).

The engine's status must be set, the engine's mix mode must be set and if auto output is not enabled, you have to enable one or more light channels with commands.

**Set channel**
Now that both the engine and the channel(s) are ready to process data, we can start sending some data. The command **ChannelSetData** can be used to set the channel to any valid value (0…255).

**See also**
With the description given here, the value will change instantly. If you want the engine to fade to the new value you should activate the fades options in the Engine Status and define some kind of fade (type and time). For more info on this refer to **Handling Engine Status**.

The light channel does not need to correspond with the DMX channel. For more info on DMX mapping and channel gain, slope limiting, and curve, refer to **Setting up the Patcher**.

### Starting a Cue List in an engine

The LanBox can store Cue Lists in its internal non-volatile memory. Any Cue List can be executed in any engine. You can start Cue-Lists at the start of the Cue List or anywhere in the Cue List.

**Initialize engine**
For now we choose engine A (default the highest priority). First, engine A need to be initialized for data output.

The engine's status must be set, the engine's mix mode must be set and if auto output is not enabled, you have to enable one or more light channels with commands.

For normal execution of a Cue List you should make sure that the Chase mode is set to 'No Chase' with the **EngineSetChaseMode** command.

The engine is now ready to handle the Cue List data.

In order to start a Cue List in an engine use the **EngineGo** command. The **EngineSetSequencing** should be active, otherwise the choosen cue is loaded, but the sequencer will not execute next step.

### Stepping through a Cue List in an engine

The LanBox can store Cue Lists in its internal non-volatile memory. Any Cue List can be executed in any engine. You can start Cue-Lists at the start of the Cue List or anywhere in the Cue List.

**Initialize engine**
First the choosen engine need to be initialized for data output. The engine's status must be set, the engine's mix mode must be set and if auto output is not enabled, you have to enable one or more light channels with commands.

In order to load a Cue List in an engine use the **EngineGo** command. Now we can step through the Cue List with the **EngineNext** and **EnginePrevious** commands.

### Creating a Cue List in an engine

The LanBox can store Cue Lists in its internal non-volatile memory. In order to create and edit a Cue List using an engine a number of steps must be taken.

**Initialize engine**
The LanBox has 8 identical engines, they only differ in priority. For now we choose engine A (default the highest priority). First engine A need to be initialized for data output. The engine's status must be set, the engine's mix mode must be set and if auto output is not enabled, you have to enable one or more light channels with commands.

For editing a Cue List you should make sure that the Chase mode is set to 'No Chase' with the **EngineSetChaseMode** command.

**Create and edit a Cue List**
The first step is to create a Cue List file in the LanBox file system. The command **CueListCreate** takes care of this and also re-enables the dynamic channel grouping feature.

The next step is to define a group, first you have to clear the Engine with the **EngineClear** command. A group consists of any number of lighting channels. To define a group just set all the channels in that group to their initial values with the **ChannelSetData** command and set the Cue Step parameters with the **EngineSetCueStepData** commands. Next use the **EngineUsesCueList** and then **EngineInsertAppendStep** command to store the initial values. Adjust the levels again for the next step and append again.

If you need to expand the group, load step 1 with the EngineLoad command, this will enable the used channels again and you can expand the group.

# LanBox

# Network, Serial & MIDI

# Commands

### Serial communication

The serial protocol for the LanBox has the same capabilities as the network and MIDI communication. This way it is possible to have total control over the LanBox through the serial port. However due to transformer isolation, network control is preferable.

The LanBox kernel has build in serial communication routines, but no protocols for transferring data. Because there is no simple way to test for or recover from transmission errors we'll chose a protocol that is simple and is rather resistant to transmission errors and can be generated by most devices that have serial communication.

The protocol will use only ASCII characters (no binary data). Each value is represented in Hexadecimal format formed by two or four ASCII characters. Each message consists of an start character, a number of values and an end character. Any deviation from this is considered a transmission error and the message is discarded.

**Prompt**
After a serial message has been received, interpreted and executed the LanBox will always transmit the > prompt. If the message was only partially received, could not be interpreted or executed a question mark ? is transmitted in stead of the prompt.

**Protocol**
Each message starts with the start-of-message character * and ends with an end-of-message character #. Reception of the start-of-message character will always result in resetting the input buffer and disposing of any received characters. Reception of the end-of-message character will start the interpretation of the message, execution of the command and reset the input buffer.

The two characters following the start-of-message character form the command number. Each character is the hexadecimal representation of one nibble of the 8 bit command number. For example command 1 would be represented as 01, command 10 as 0a, command 210 as d2.

The command number can be followed by any number of parameters. Each parameter can be a 8 bit value (2 hex characters) or a 16 bit value (4 hex characters). The type and amount of values depend on the command number. It is legal for a command to have no parameters. For example the following command is legal *05#. The only restriction is that the total number of parameters must be smaller than 250 bytes (1500 in LCX & LCM).

**8 or 16 bit address mode**
As the new LCE, LCM and LCX boxes goes beyond 255 light channels, it means that a channel should be a 16 bit value (High byte first). For the LCE, LCX and LCM there is a command **Common16BitMode** (65 hex ) in order to set the box so it will accept 16 bit channel addresses.

### MIDI communication

**Channels**
Each MIDI connection consists of 16 independent MIDI channels. In the LanBox it is possible to assign each pair of MIDI channels to an engine. There are 8 engines so each engine can have its own MIDI pair.

It is required to have a pair of MIDI channels for an engine, because the range of Note-Ons in one MIDI channel (0-127) is not enough to access all light channels in an engine (1-250).

**Reception and transmission**
The LanBox is able receive and transmit MIDI. Transmission of MIDI will only be initiated after a request for data. For example it is possible to have the LanBox dump all values of the light channels of the mixer as a SysEx dump as a response to a dump request.

**Protocol**
Manipulation of the value of light channels in engines is done by MIDI note-on messages. It is possible to ignore all Note-Ons with a velocity of 0 and Note-Offs, temporary or permanently.

Most other simple editing commands (start, stop, halt, next,... the ones that have just one parameter) is implemented as controller messages.

**MIDI SysEx commands**
All ASCII commands are also implemented as SysEx messages. The request is:
F0 00 20 40 <Dev ID> 50 <Master ID> <* cmd #> F7
The LanBox responds with:
F0 00 20 40 <Dev ID> 51 <Master ID> <* data # > 3E F7
or
F0 00 20 40 <Dev ID> 51 <Master ID> <* data # > 3F F7
for invalid commands

**MIDI Show Control**
It is possible to control the LanBox with MIDI Show Control (MSC). The purpose of MIDI Show Control is to allow MIDI systems to communicate with and to control dedicated intelligent control equipment in theatrical, live performance, multi media, audio-visual and similar environment.

The implementation of MSC is consistent with the recommended minimum set 2 (No time code, full data capability).

In order to receive MSC command each engine should have its own MSC device_ID. When using more than one LanBox in a MSC system, each engine should have a unique device_ID. This ID is stored in non-volatile memory and can be changed with the EngineSetDeviceID command.

**Index of Commands**



**Index of Commands**

**ChannelSetData**

The LanBox has multiple engines. Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine can control up to 250/512 light channels.

The 'ChannelSetData' command can set the value of any number of light channels within one engine.

The serial command number for setting the value of a light channel in an engine is 201. After this command one byte is reserved for the target engine number, next any number of channel number, channel value pairs (both eight bit values) can be placed.

General form
* C9 ee nn vv nn vv…nn vv #

| | | |
|---|---|---|
| ee | Engine number | (1…8) |
| nn | Channel number | (0…250) |
| vv | Channel value | (0…255) |

Note: nn becomes nnnn when in 16 bit address mode!

For example

* C9 02 03 04 #
Sets in engine 2
light channel 3 to a value of 4.

* C9 02 01 22 02 33 04 77 #
Sets in engine 2
light channel 1 to a value of 34 (22h)
light channel 2 to a value of 51 (33h)
light channel 4 to a a value of 119 (77h)

* C9 02 00 00 #
Clear all channels in engine B

**ChannelSetData**

The MIDI method for setting the value of a light channel in an engine is using a Note-On message (144…159). The MIDI channel on which the note-on is send determines the engine in which the channel must be set. The note-pitch determines the channel number and the note-velocity determines the value.

Note that because the the note-velocity only has a 0-127 range, the velocity is multiplied by two in order to get the light channel range. note-velocity 127 is mapped to DMX output value of 255 (in stead of 254) in order to be able to set a channel full on.

The note-pitch also has only a 0-127 range, so it is not possible to map all DMX channels directly to MIDI channels. In order to have total control from MIDI over all DMX channels it is possible to assign two MIDI channels to an engine, a primary and a secondary channel. This way we have sufficient MIDI channels. The MIDI channel are mapped to the DMX channels in the following way:

| Primary MIDI ch note pitch | Secondary MIDI ch note pitch | DMX channel address |
|---|---|---|
| | | |
| 0 | - | - |
| 1 | - | 1 |
| … | … | … |
| 127 | - | 127 |
| - | 0 | 128 |
| - | 1 | 129 |
| … | … | … |
| - | 122 | 250 |
| | | |

In order to allow direct input from a MIDI keyboard, without a need to program, the Note-On messages have a auto output function that is switched on by default. When a Note-On message is received for a certain light channel (a certain pitch), that channel is enabled automatically.

If the auto output function is not desired, it can be switched off with the EngineSetAutoOutput command.

When a EngineSetSustain command has activated the Sustain mode, all Note-Ons with a velocity of 0 and all note-off messages are not executed, but stored in the sustain buffer, until the sustain mode is turned off. If a note-on with a velocity > 0 is received the note-off message is removed from the sustain buffer. When the sustain mode is turned off all Note-Offs in the sustain buffer are executed. When a EngineSetIgnoreNoteOff command has activated the Ignore mode, all note-off messages is ignored (also the note-on messages with a velocity of 0).

**EngineSetSustain (MIDI)**

The LanBox has multiple engines.Each engine can be used for sequencing Duelists and direct control of light channels. Each engine can control up to 250 light channels. The EngineSetSustain On command defers the execution of note-on messages with a velocity of 0 and note_off messages until a EngineSetSustain Off command is received.

The EngineSetSustain command is implemented for MIDI as Controller message number 64. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). If the value of the controller <=63, then the sustain status is off. If the value of the controller >= 64, then the sustain status status is on.

**EngineSetIgnoreNoteOff (MIDI)**

The LanBox has multiple engines.Each engine can be used for sequencing Duelists and direct control of light channels. Each engine can control up to 250 light channels. The EngineSetIgnoreNoteOff On command ignores all note_off messages until a EngineSetSustain Off command is received.

The EngineSetIgnoreNoteOff command is implemented for MIDI as Controller message number 65. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). If the value of the controller <=63, then the ignore status is off. If the value of the controller >= 64, then the ignore status status is on.

When Controller 65 is set to >=64 then:
All Note-Offs AND Note-On with a velocity of zero are ignored. AND A Note-On with a velocity of 1 will be translated to DMX value zero.

Note:
A Note-On with a velocity of 2..126 will be translated * 2 to DMX value. This means DMX value of 2 can not be made (just as 254 and all odd values).

**ChannelSetOutputEnable**

The LanBox has multiple engines. Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine can control up to 250/512 light channels. A channel has three status attributes, Output, Edit and Solo. Output determines if the channel is written to the mixer. Edit determines if the channel is written to a cue list scene. Solo determines if a channel is in solo mode.

The 'ChannelSetOutputEnable' command can set the output attribute of any number of light channels within one engine.

The serial command number for setting the output attribute of a light channel in an engine is 202. After this command one byte is reserved for the target engine number, next any number of channel number, enable status pairs (both eight bit values) can be placed.

General form
* CA ee nn aa nn aa…nn aa #
ee Engine number (1…8)
nn Channel number (0…250)
aa Channel output (>0 = Enabled, 0 = Disabled)

Note: nn becomes nnnn when in 16 bit address mode!

For example

* CA 02 03 FF #
Sets in engine 2
light channel 3 is enabled

* CA 02 01 FF 02 00 04 FF #
Sets in engine 2
light channel 1 is enabled
light channel 2 is disabled
light channel 4 is enabled

* CA 02 00 00 #
Disable all channels in engine B

**ChannelSetOutputEnable**

In MIDI this command is implemented only as SysEx commands, see MIDI SysEx Commands.

**ChannelSetEditEnable**

The LanBox has multiple engines. Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine can control up to 250/512 light channels. A channel has three status attributes, Output, Edit and Solo. Output determines if the channel is written to the mixer. Edit determines if the channel is written to a cue list scene. Solo determines if a channel is in solo mode.

The ‘ChannelSetEditEnable’ command can set the output attribute of any number of light channels within one engine.

The serial command number for setting the edit attribute of a light channel in an engine is 204. After this command one byte is reserved for the target engine number, next any number of channel number, enable status pairs (both eight bit values) can be placed.

General form
* CC ee nn aa nn aa…nn aa #
ee Engine number (1…8)
nn Channel number (0…250)
aa Channel Edit (>0 = Enabled, 0 = Disabled)

Note: nn becomes nnnn when in 16 bit address mode!

For example

* CC 02 03 FF #
Sets in engine 2
light channel 3 is Edit enabled

* CC 02 01 FF 02 00 04 FF #
Sets in engine 2
light channel 1 is Edit enabled
light channel 2 is Edit disabled
light channel 4 is Edit enabled

* CC 02 00 00 #
Disable all Edit channels in engine B

**ChannelSetEditEnable**

In MIDI this command is implemented only as SysEx commands, see MIDI SysEx Commands.

**ChannelSetSolo**

The LanBox has multiple engines. Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine can control up to 250/512 light channels. A channel has three status attributes, Output, Edit and Solo. Output determines if the channel is written to the mixer. Edit determines if the channel is written to a cue list scene. Solo determines if a channel is in solo mode.

The ‘ChannelSetSolo’ command can set the solo attribute of any number of light channels within one engine.

The serial command number for setting the enable attribute of a light channel in an engine is 203. After this command one byte is reserved for the target engine number, next any number of channel number, solo status pairs (both eight bit values) can be placed.

General form
* CB ee nn aa nn aa…nn aa #

| | |
|---|---|
| ee | Engine number (1…8) |
| nn | Channel number (0…250) |
| aa | Solo enable (>0 = Enabled, 0 = Disabled) |

Note: nn becomes nnnn when in 16 bit address mode!

For example

* CB 02 03 FF #
Sets in engine 2
light channel 3 is set to solo mode

* CB 02 01 FF 02 00 04 FF #
Sets in engine 2
light channel 1 is set to solo mode
light channel 2 is removed from solo mode
light channel 4 is set to solo mode

* CB 02 00 FF #
Solo all channel in engine 2

**ChannelSetSolo (MIDI)**

In MIDI this command is implemented only as SysEx commands, see MIDI SysEx Commands.

**ChannelReadData**

The LanBox has multiple engines. Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine can control up to 250/512 light channels.

The ‘ChannelReadData’ command can return the value of any number of light channels within one engine.

The serial command number for setting the value of a light channel in an engine is 205. After this command one byte is reserved for the target engine number, followed by the first channel and the number of channels that must be returned.

General form
* CD ee ffff nn #
ee Engine number (1..8)*
on LC+ 10=DMXoutbuf, 9=mixbuf
on LCM 255=DMXoutbuf, 254=mixbuf, 253=extinp
on LCX 255=DMXoutbuf, 254=mixbuf, 253=extinp and 252=DMXinBuf
ffff First Channel number to be returned (high byte First)
nn Number of channels to be returned

Note: ff becomes ffff when in 16 bit address mode!
Note: 9=Mixer data, 10=DMX output data on LC
Note: 255=DMX output data , 254=Mixer data, 253=ExtIn data, 252= DMX input data.

For example

* CD 02 03 02 #
Returns from engine 2
light channels 3 and 4.
* aa bb #>

* CD 02 01 D6 #
Returns from engine 2
light channels 1 through 213
* aa bb … zz # >

**ChannelReadData (MIDI)**

In MIDI this command is implemented only as SysEx commands, see MIDI SysEx Commands.

**ChannelReadStatus**

The LanBox has multiple engines. Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine can control up to 250/512 light channels. A channel has four status attributes, Output, Enabled, Solo and Fader. Output determines if the data is fed to the mixer. Only enabled channels can be stored into a cue step's scene. Solo determines if a channel is in solo mode. Fader shows if a fade is going on that channel.

The 'ChannelReadStatus' command can return the attributes of any number of light channels within one engine.

The serial command number for returning the status attributes of a light channel in an engine is 206 (CEh). After this command one byte is reserved for the target engine number, followed by the first channel and the number of channels that must be returned.

The output status is returned in bit 0 (the least significant bit), The enable status is returned in bit 1, the solo status is returned in bit 2 and and the fader status is returned in bit 3.

General form
* CE ee ff nn #

| | |
|---|---|
| ee | Engine number |
| ff | First Channel number to be returned |
| nn | Number of channels to be returned |

For example

* CE 02 03 04 #
Returns from engine 2
enable mode for light channels 3, 4, 5, 6
* 00 0B 04 05 # >

| Channel | Output | Enable | Solo | Fader |
|---|---|---|---|---|
| 3 | - | - | - | - |
| 4 | x | x | - | x |
| 5 | - | - | x | - |
| 6 | x | - | x | - |

* CE 02 01 FA #
Returns from engine 2
enable mode for light channels 1 through 250
* 00 00 00 01 01 00 05 00 00 00 01 01 05 01 … 00 01 # >

**ChannelReadStatus**

In MIDI this command is implemented only as SysEx commands, see MIDI SysEx Commands.

**EngineSetOutput**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a status that determines the way the engine operates. The output attribute determines if the engine writes its data to the mixer or not. The EngineSetOuput command sets or clears the output status.

The serial command number for setting the output status is 72 (48h hexadecimal). The command has two 8-bit parameters, the engine number for which the output status must be set, the value for the output (0 -> Off, >0 -> On)

General form
* 48 ee vv #

| | |
|---|---|
| ee | Engine number (1…8) |
| vv | Output status (0 -> Off, >0 -> On) |

For example

* 48 01 FF #
Set in engine A (01) the output status on (255).
* 48 02 00 #
Set in engine B (02) the output status off (0).

**EngineSetOutput**

The ChannelSetOutputEnable command is implemented for MIDI as Controller message number 72. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). If the value of the controller <=63, then the output status is off. If the value of the controller >= 64, then the output status is on.

**EngineSetSequencing**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a status that determines the way the engine operates. Sequencing determines if the engine is processing any Cue Lists. The EngineSetSequencing command sets or clears the sequencing status.

The serial command number for setting the sequencing status is 73 (49h hexadecimal). The command has two 8-bit parameters, the engine number for which the sequencing status must be set, the value for the status (0 -> Off, >0 -> On)

General form
* 49 ee vv #

| | |
|---|---|
| ee | Engine number (1…8) |
| vv | Sequencing status (0 -> Off, >0 -> On) |

For example

* 49 01 FF #
Set in engine A (01) the sequencing status on (255).
* 49 02 00 #
Set in engine B (02) the sequencing status off (0).

**EngineSetSequencing**

The EngineSetSequencing command is implemented for MIDI as Controller message number 73. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). If the value of the controller <=63, then the sequencing status is off. If the value of the controller >= 64, then the sequencing status is on.

**EngineSetFading**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a status that determines the way the engine operates. Fading determines if the engine executes fades when a channel values changes. The EngineSetFading command sets or clears the fading status.

The serial command number for setting the fading status is 70 (46h hexadecimal). The command has two 8-bit parameters, the engine number for which the fading status must be set, the value for the status (0 -> Off, >0 -> On)

General form
* 46 ee vv #

| | |
|---|---|
| ee | Engine number (1…8) |
| vv | Fading status (0 -> Off, >0 -> On) |

For example

* 46 01 FF #
Set in engine A (01) the fading status on (255).
* 46 02 00 #
Set in engine B (02) the fading status off (0).

**EngineSetFading**

The EngineSetFading command is implemented for MIDI as Controller message number 70. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). If the value of the controller <=63, then the fading status is off. If the value of the controller >= 64, then the fading status is on.

**EngineSetSolo**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a status that determines the way the engine operates. Solo determines if the engine is in solo mode. The EngineSetSolo command sets or clears the solo status.

The serial command number for setting the engine solo status is 74 (4Ah hexadecimal). The command has two 8-bit parameters, the engine number for which the solo status must be set, the value for the status (0 -> Off, >0 -> On)

General form
* 4A ee vv #

| | |
|---|---|
| ee | Engine number (1…8) |
| vv | Fading status (0 -> Off, >0 -> On) |

For example

* 4A 01 FF #
Set in engine A (01) the solo status on (255).
* 4A 02 00 #
Set in engine B (02) the solo status off (0).

**EngineSetSolo**

The EngineSetSolo command is implemented for MIDI as Controller message number 74. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). If the value of the controller <=63, then the solo status is off. If the value of the controller >= 64, then the solo status is on.

**SetEngineMixMode**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a mix mode that determines the way the engine mixes its data with the other engines. The EngineSetMixMode command sets the mix mode.

There are four mix modes that can be set Copy (1), Mix-Up (2), Mix-Down (3) and Transparent(4). During mix (which occurs 20 times per second, just before the DMX frame is transmitted) the channels of all the engines are mixed together. Because it is legal to have two engines work on the same channel, there must be a way to determine how the values are mixed.

The engines are processed in order of priority, starting with lowest priority engine H (or 8). If an engine is in Copy mode, the value of each active channel in that engine is simply copied to the mixer (overwriting data of lower priority engines). If an engine is in Mix-up mode, it only copies active channel values that are higher than the current mixer value. If an engine is in Mix-down mode, it only copies active channel values that are lower than the current mixer value. If an engine is in transparent mode will will average channels value with the mixer values according to the setting of the Transparency depth.

The serial command number for setting the engine solo status is 71 (47h hexadecimal). The command has two 8-bit parameters, the engine number for which the mix mode must be set, the value for the status (0=No Output, 1=Copy, 2=Mix-Up, 3=Mix-Down, 4=Transparent)

General form
* 47 ee mm #
ee Engine number
mm Mix mode (0=No Output, 1=Copy, 2=Mix-Up, 3=Mix-Down, 4=Transparent)

For example

* 47 01 03 #
Set the Mix-Down (3) mode in engine A (01)

**EngineSetMixMode**

The EngineSetMixMode command is implemented for MIDI as Controller message number 71. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). If the value of the controller is 0, then the mix mode is Copy. If the value of the controller is 1, then the mix mode is Mix-Up. If the value of the controller is 2, then the mix mode is Mix-Down. A value of 3, results in transparent mode.

**EngineSetChaseMode**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a chase mode that determines the way the engine sequences Cue Lists. The EngineSetChaseMode command sets the chase mode mode.

There are 9 chase modes

- No Chase (0)
- Chase Up (1)
- Chase Up Repeated (2)
- Chase Down (3)
- Chase Down Repeated (4)
- Chase Random (5
- Chase Random Repeated (6)
- Chase Bounce (7)
- Chase Bounce Repeated (8)

A chase determines the way a Cue List is executed. When the chase mode is set to No Chase, the Cue List is executed only once, from start to end and with the defined timing. With every other chase, the timing depends on the chase speed and the sequence of the Cue List depends on the chase type.

The serial command number for setting the engine solo status is 75 (4Bh hexadecimal). The command has two 8-bit parameters, the engine number for which the chase mode must be set, the chase mode (0…8)

General form

* 4B ee mm #

| | |
|---|---|
| ee | Engine number |
| mm | Chase mode (0…8) |

For example

* 4B 01 03 #

Set the Chase Down (3) mode in engine A (01)

**EngineSetChaseMode**

The EngineSetChaseMode command is implemented for MIDI as Controller message number 75. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). The value of the controller should correspond with one of the chase modes.

**EngineSetChaseSpeed**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a chase mode that determines the way the engine sequences Cue Lists. A chase is executed with a certain speed. The EngineSetChaseSpeed command sets this chase speed.

The Chase speed parameter ranges from 255..0, where 127 is normal, 0 is half speed, 64 double speed 32 speed*4, etc

The serial command number for setting the engine chase speed is 76 (4Ch hexadecimal). The command has two 8-bit parameters, the engine number for which the chase mode speed be set, the value for the speed (0..255)

General form
* 4C ee ss #

| | |
|---|---|
| ee | Engine number |
| ss | Chase speed (255..0) |

For example

* 4C 01 ?? #
Set the Chase speed ? (?) in engine A (01)

**EngineSetChaseSpeed**

The EngineSetChaseSpeed command is implemented for MIDI as Controller message number 76 The MIDI channel determines which engine should process the command (depending on the MIDI mapping). The value is the chase speed.

**EngineSetFadeType**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a fade type attribute that determines the way a change in light channel value is handled. The EngineSetFadeType command sets this fade type speed.

There are 7 different fade types

- No Fade (0)
- Fade-In (1)
- Fade-Out (2)
- Cross Fade (3)
- No Fade (4)
- Fade-In with constant rate (5)
- Fade-Out with constant rate (6)
- Cross Fade with constant rate (7)

'No Fade' (0) allows an abrupt change of light channel values. With a 'Fade-In' (1), value that increase are gradually changed (speed depending on the fade time), while values that decrease are set instantly. With a 'Fade-Out' (2), value that decrease are gradually changed (speed depending on the fade time), while values that increase are set instantly. With a 'Cross Fade' (3) all value changes are gradual (speed depending on the fade time).

The variants with constant rate ensure that the least significant byte (LSB) of a 16 bit fader will be incremented with a constant rate. This should result in smoother fades on scanners, but also in a less accurate fade time.

The serial command number for setting the engine fade type is 77 (4Dh hexadecimal). The command has two 8-bit parameters, the engine number for which the fade type must be set, the fade type (0…3)

General form

* 4D ee tt #

| | |
|---|---|
| ee | Engine number |
| tt | Fade type (0…7) |

For example

* 4D 01 02 #

Set the Fade type Fade-Out (2) in engine A (01)

**EngineSetFadeType**

The EngineSetFadeType command is implemented for MIDI as Controller message number 77 The MIDI channel determines which engine should process the command (depending on the MIDI mapping). The value of the controller should correspond with one of the fade types.

**EngineSetFadeTime**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a fade type attribute that determines the way a change in light channel value is handled. Each fade uses a fade time parameter. The EngineSetFadeTime command sets this fade time.

The fade time parameter is encoded through the 'LanBox-LC + Time Encoding Table'.

The serial command number for setting the engine fade time is 78 (4Eh hexadecimal). The command has two 8-bit parameters, the engine number for which the fade time must be set, the fade time (0…91)

General form
* 4E ee tt #

| | |
|---|---|
| ee | Engine number |
| tt | Fade time (0…91) |

For example

* 4E 01 1C #
Set the Fade time of engine A (01) to 2.2 seconds (1Ch = 28)

**EngineSetFadeTime**

The EngineSetFadeTime command is implemented for MIDI as Controller message number 78. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). The value of the controller should be the coded Fade time.

**EngineSetTransparencyDepth**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. Each engine has a Transparency Depth attribute that determines the way transparency mix mode operates. The EngineSetTransparencyDepth command sets this depth.

The transparency depth is a value ranging from 0 to 255 that determines how much of the engines value and how much of the mixer value should be used for the resulting output. With a transparency depth of 255, only the mixer value is used, with a transparency depth of 0 only the engine value is used.

The serial command number for setting the engine fade time is 99 (63h hexadecimal). The command has two 8-bit parameters, the engine number for which the Transparency Depth must be set and the depth (0…255)

General form
* 63 ee dd #

| | | | |
|---|---|---|---|
| ee | Engine number | (1…8) | |
| dd | Depth | | (0…255) |

For example

* 63 01 1C #
Set the transparency depth of engine A (01) to 28 (about 11%)

**EngineSetTransparencyDepth**

The EngineSetTransparencyDepth command is implemented for MIDI as Controller message number 99. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). The value of the controller should be the transparency depth.

**EngineGetStatus**

The LanBox has multiple engines.Each engine maintains a number of parameters that determine the way the engine operates. With the EngineGetStatus command we can collect these parameters from the LanBox.

The serial command number for getting the engine status is 10 (0Ah hexadecimal). The command has one 8-bit parameter, the engine number. The LanBox will respond with a dump of all parameters of that engine, followed by the prompt

General form
* 0A ee#
ee Engine number (1…8)

Response
* os ss fs ls ms sr sh hhhh cccc ss cm cs ft fm rs rmrm td li ps id ae st s1 s2 s3 s4 s5 s6 wa #

| | |
|---|---|
| os | Output Status (0 -> Off, >0 -> On) |
| ss | Sequence Status (bit 0..7 = Eng A..H) |
| fs | Fade Status (0 -> Off, >0 -> On) |
| ls | Solo Status (0 -> Off, >0 -> On) |
| ms | Mix Status (1=Copy, 2=Up, 3=Down, 4=Transp) |
| sr | Current Scene Relative (0 -> Off, >0 -> On) |
| sh | Current duration time ( Time Encoding) |
| hhhh | Remaining duration time (units of 50 ms) |
| cccc | Current Cue List (1…500) |
| cs | Current Cue Step (1…100) |
| cm | Current Chase mode (0, 1, 2, 3, 4, 17, 18, 19, 20) |
| cs | Current Chase speed (or faders speed) |
| ft | Manual Fade type (0…7) |
| fm | Manual Fade time (LanBox-LC + Time Encoding) |
| rs | Remaining Fade time (LC + Time Encoding) |
| rmrm | Remaining Fade time (units of 50 ms) |
| td | Transparency Depth |
| li | Loading Indication (idle in ms) |
| ps | Pause Status (bit 0..7 = Eng A..H) |
| id | Device ID (0…127) |
| ae | Auto Enable Status (0 -> Off, >0 -> On) |
| st | Current Step Type (type of the current Duelist) |
| s1 | Step data 1 (depends on Step Type) |
| s2 | Step data 2 (depends on Step Type) |
| s3 | Step data 3 (depends on Step Type) |
| s4 | Step data 4 (depends on Step Type) |
| s5 | Step data 5 (depends on Step Type) |
| s6 | Step data 6 (depends on Step Type) |
| wa | Engine is waiting (bit 0..7 = Eng A..H) |

**EngineGetStatus**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**EngineGetStatus(cont)**

**EngineGetStatus(cont)**

For example

* 0A 01# Get the parameters of engine A (01)

* FF FF FF 00 02 FF 3F 02 30 01 2C 04 00 00 03 1C 1D 00 24 55 20 00 1E FF 01 03 1F 2C 08 93 40 00 # >

| | |
|---|---|
| FF | Output status is On |
| 03 | Eng A & B are sequencing |
| FF | Fade Status is On |
| 00 | Solo Status is Off |
| 02 | Mix mode is Mix-Up |
| FF | Current Scene is Relative |
| 3F | Current duration time is 1 minute* |
| 0230 | Remaining duration time is 28 sec* |
| 012C | Current Cue List is 300 (012C = 300) |
| 04 | Current Cue Step is 4 |
| 00 | Current Chase Mode is No Chase |
| 00 | Current Chase speed is 0 |
| 03 | Manual Fade type is Crucified |
| 1C | Manual Fade time is 2.2 seconds* |
| 1D | Remaining Fade time start is 2.4 seconds* |
| 0024 | Remaining Fade time is 1.8 seconds* |
| 55 | Transparency Depth is 33% (55h is 85 is 33%) |
| 20 | Loading Indication is 32 |
| 01 | Engine A is in Pause |
| 1E | Device ID is set to 30 |
| FF | Auto Enable Status is On |
| 01 | Current Cue Step is a Cuisine |
| 03 | Fade type for current step is Crucified |
| 1F | Fade time for current step is 3.0 seconds * |
| 2C | Duration time for current step is 10 seconds * |
| 08 | Pointer to Scene |
| 93 | Pointer to Scene |
| 40 | Pointer to Scene |
| 00 | No Engine is waiting on a Go. |

* See Time Encoding table

**EngineGetGlobalData**

Besides engine data the LanBox has some global data that may be of interest to the user. With the command EngineGetGlobalData this information may be retrieved.

The serial command number for getting the engine status is 11 (0Bh hexadecimal). The command has no parameters. The LanBox will respond with a dump of all global parameters, followed by the prompt.

General form
* 0B #

Response
* br do dc ns nnnn sx#

| | |
|---|---|
| br | Baud rate (0=38400, 1=19200, 2=9600, 3=MIDI) |
| do | DMX offset (0…255) |
| dc | Number of DMX channels (0…250) |
| ns | Network name size (0…13) |
| nnnn | Network name (fixed 13 chars) |
| sx | SysEx ID |

In 16 bit mode the response is much longer, and different:
* br do dc ns nnnn sx ipa ipm ipg dl dd ds dc ul ua up#

| | |
|---|---|
| br | Baud rate (0=38400, 1=19200, 2=9600, 3=MIDI) |
| do | DMX offset (0…65535) |
| dc | Number of DMX channels (0…65535) |
| ns | Network name size (0…13) |
| nnnn | Network name (fixed 13 chars) |
| sx | SysEx ID |
| ipa | IP address (4 bytes HBF) |
| ipm | IP mask (4 bytes HBF) |
| ipg | IP gateway (4 bytes HBF |
| dl | DMX input dest layer (0-255) |
| dd | DMX input dest offset (0-65535) |
| ds | DMX input src offset (0-65535) |
| dc | DMX input size (0-65535) |
| ul | UDP in dest layer (0=disable, 254=mixer) |
| uia | UDP in src IP ( 4 byes HBF) |
| uip | UDP in rx port (0-65535) |
| uid | UDP in dest start channel (0-65535) |
| uis | UDP in src start channel (0-65535) |
| uic | UDP in channel count (0-65535) |
| uop | UDP out port (0-65535) |
| uod | UDP out dest start channel (0-65535) |
| uos | UDP out src start channel (0-65535) |
| uoc | UDP out channel count (0-65535) |
| uob | UDP out bits (1 = dmxout, 2 = mixer, 4 = inputs, 8 = dmxin) |

**EngineGetGlobalData**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**EngineUsesCueList**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists, direct control of light channels and creation and editing of Cue Lists and Cue Steps. The **EngineUsesCueList** command is used with editing Cue Lists.

For editing existing Cue Lists the Cue List will most often be loaded with the **EngineLoad** command, so the engine knows which Cue List is the target. In cases where loading is not desired (because it would destroy all edited parameters) or not possible (because the Cue List is empty), the **EngineUsesCueList** command can be used to tell the engine which Cue List should be the target.

The **EngineUsesCueList** command determine how commands like **EngineInsertAppendStep** and **EngineReplaceStep** work.

The serial command number for setting the engine Cue List is 12 (0Ch hexadecimal). The command has one 8-bit parameter, the engine number for which the Cue List must be set and one 16 bit parameter, the Cue List number (1…500)

General form
* 0c ee cc cc #

| | | |
|---|---|---|
| ee | Engine number | (1…8) |
| cccc | Cue List num | (1…500) |

For example

* 0c 01 01 23 #
Tells engine A to use Cue List 291 (0123h)

**EngineUsesCueList**

The **EngineUsesCueList** command is implemented for MIDI as Controller message number 21 (this is a 14 bit controller, 21 is the MSB, controller 53 is the LSB). The MIDI channel on which the controller is send determines which engine's ID is set, the 14-bit value of the controller is the Cue List number which must be used.

**EngineSetCueStepType**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists, direct control of light channels and creation and editing of Cue Lists and Cue Steps. The EngineSetCueStepType command can be used to edit the type of a Cue Step.

Each Cue Step in a Cue List has a type. Known types are

1) Cue Scene
2) Cue Refrence Scene
10) Start Engine
11) Stop Engine
12) Suspend Engine
13) Resume Engine
14) Start Remote Engine
15) Stop Remote Engine
20) Goto Cue Step
21) Go Next
22) Go Previous
23) Loop To
24) Wait Engine
Wait SMPTE
Wait Pulse
30) Set Engine Attributes
31) SetEngineMixMode
32) Set Engine Chase

Each Cue Step type has a set of parameters. These parameters can be set with the EngineSetCueStepx commands. For a detailed description of the Cue Step types and their parameters please refer to the Cue List description

The serial command number for EngineSetCueStepType is 79 (4Fh hexadecimal). The command has two 8-bit parameters. The first parametsr is the engine number the second is the Cue List type.

General form
* 4F ee tt #
ee Engine number (1…8)
tt Cue List type

For example

* 4F 01 01 #
Defines current Cue Step in engine A as a Cue Scene

* 4F 01 0A #
Defines current Cue Step in engine A as a Start Engine

**EngineSetCueStepType**

The EngineSetCueStepType command is implemented for MIDI as Controller message number 79. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). The value of the controller is the type of a Cue Step.

**EngineSetCueStep1** … **EngineSetCueStep6**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists, direct control of light channels and creation and editing of Cue Lists and Cue Steps. The EngineSetCueStepx command can be used to edit the step data of a Cue Step. For a detailed description of the Cue Step types and their parameters please refer to the Cue List description

The serial command numbers for EngineSetCueStep are 80…85 (50h…55h hexadecimal). Each command has two 8-bit parameters. The first parametsr is the engine number the second is the Cue List data.

The meaning of the contents of the Cue Step data depends on the Cue Step type which can be set with the EngineSetCueStepType command.

General form
* 5x ee dd #

| | |
|---|---|
| ee | Engine number (1…8) |
| dd | Cue List data |

For example

* 50 01 02 #
Sets CueStep1 data to 2

* 52 01 0A #
Sets CueStep2 data to 10

**EngineSetCueStep1** … **EngineSetCueStep6**

The EngineSetCueStepx command is implemented for MIDI as Controllers message number 80..85. The MIDI channel determines which engine should process the command (depending on the MIDI mapping). The value of the controller should be the Cue Step data.

**EngineGo**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. The EngineGo command can be used to start execution of a Cue List in an Engine.

The serial command number for EngineGo is 86 (56h hexadecimal). The command has one 16-bit and two 8-bit parameters. The first parametsr is the engine number in which the Cue List must be started. The second is the Cue List number. The third is the Cue Step number. The Cue Step number is optional. If not supplied the LanBox will start the Cue List at Step 1.

General form
* 56 ee cccc [ss] #

| | |
|---|---|
| ee | Engine number (1…8) |
| cccc | Cue List number (1…500) |
| ss | Cue Step number (1…100) |

For example

* 56 01 01 2C 04 #
Start Cue List 300 in engine A (01) at Cue Step 4

* 56 01 01 2C #
Start Cue List 300 in engine A (01) at Cue Step 1

**EngineGo**

The EngineGo command is implemented for MIDI as Controller message number 20 (this is a 14 bit controller, 20 is the MSB, controller 52 is the LSB). The MIDI channel on which the controller is send determines which engine's ID is set, the 14-bit value of the controller is the Cue List number which must be started.

The LanBox expects to receive the MSB (controller 20) first and will not Go the engine until the LSB (controller 52) is received. The MSB is remembered and used for any subsequent received LSBs.

For example to Go Cue List 200:
Controller 20 with a value of 1
Controller 52 with a value of 72 -> Triggers go!

Next Go Cue List 220
Controller 52 with a value of 92 -> This will trigger the start!

If the Cue List should not be started at the first Cue Step (which is the case described above), another method should be used. First use controller 21 (this is a 14 bit controller, 21 is the MSB, controller 53 is the LSB), then use controller 86 to set the Cue Step number and to execute the start command.

The LanBox expects to receive the MSB (controller 21) first and then the LSB (controller 53) is received and will not start the engine until the controller 86 is received.

For example to start Cue List 200 step 5
Controller 21 with a value of 1
Controller 53 with a value of 72
Controller 86 with a value of 5 -> Trigger the Go!

Next start Cue List 220 (step 0)
Controller 52 with a value of 92 -> Triggers Go!

Next step Cue List 220 at step 10
Controller 86 with a value of 10 -> Triggers Go!

**Note: Keep the 2 or 3 controller messages tight together, if you are using multiple MIDI channels.**

For reasons of compatibility with the LanBox-LC II, it's also possible to start an engine with a Program change message. For example Sending a program change with value 30 on a MIDI channel assigned to engine A will start Cue List 30 in Engine A. As with the LanBox-LC II, a bank select controller (00) can be used to set a so called base number. This base value is multiplied with 128 and then added to the Program change value, so it becomes possible to start Cue List with a higher number than 127.

**EngineClear**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. The EngineClear command can be used to stop execution of a Cue List in an Engine.

The **EngineClear** command will clear the Sequencing attribute of the engine, disable all channels of that engine and reinitialize each channel value in that engine to 0. The current Cue List number and current Cue Step number are cleared.

The serial command number for EngineClear is 87 (57h hexadecimal). The command has one 8-bit parameter, the engine number which must be stopped.

General form
* 57 ee #
ee Engine number (1…8)

For example

* 57 01 #
Stops Cue List execution in engine A (01)

**EngineClear**

The EngineClear command is implemented for MIDI as Controller message number 87. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the not used.

**EnginePauze**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. The EnginePauze command can be used to suspend execution of a Cue List in an Engine. The engine will keep outputting the current values, but the faders will stop and remaining duration time is frozen.

Execution can be resumed with the EngineResume command. An EngineGo command will also put the engine out of Halted mode and will also kill all suspended faders.

The serial command number for EnginePauze is 88 (58h hexadecimal). The command has one 8-bit parameter, the engine number which must be halted.

General form
* 58 ee #
ee Engine number (1…8)

For example

* 58 01 #
Halts Cue List execution in engine A (01)

**EnginePauze**

The EnginePauze command is implemented for MIDI as Controller message number 88 The MIDI channel on which the controller is send determines which engine’s ID is set, the value of the controller is the not used.

**EngineResume**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. The EngineResume command can be used to resume execution of a Cue List in an Engine that was halted with the EnginePauze command. EngineResume will resume exactly at the point where executions was halted. Faders will continue and the Sequencing attribute is set On again. EngineResume has no effect if the engine was not halted.

The serial command number for EngineResume is 89 (59h hexadecimal). The command has one 8-bit parameter, the engine number which must be halted.

General form
* 59 ee #
ee Engine number (1…8)

For example

* 59 01 #
Resumes Cue List execution in engine A (01)

**EngineResume**

The EngineResume command is implemented for MIDI as Controller message number 89 The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the not used.

**EngineNext**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. The EngineNext command can be used to proceed execution of a Cue List in an Engine with the next Cue Step in that Cue List. EngineNext will not alter any attributes nor put the engine in or out of halt mode.

The serial command number for EngineNext is 90 (5Ah hexadecimal). The command has one 8-bit parameter, the engine number.

General form
* 5A ee #
ee Engine number (1…8)

For example

* 5A 01 #
Proceed with next Cue Step in current Cue List in engine A (01)

**EngineNext**

The EngineNext command is implemented for MIDI as Controller message number 90. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the not used.

Controller 117 or 118 are also supported (both work equally and are supported for compatibility reasons).If the value if the controller is lower than 64, the Engine Next is executed, otherwise EnginePrevious.

**EnginePrevious**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists and direct control of light channels. The EnginePrevious command can be used to proceed execution of a Cue List in an Engine with the previous Cue Step in that Cue List. EnginePrevious will not alter any attributes nor put the engine in or out of halt mode.

The serial command number for EnginePrevious is 91 (5Bh hexadecimal). The command has one 8-bit parameter, the engine number.

General form
* 5B ee #
ee Engine number (1…8)

For example

* 5B 01 #
Proceed with previous Cue Step in current Cue List in engine A (01)

**EnginePrevious**

The EnginePrevious command is implemented for MIDI as Controller message number 91. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the not used.

Controller 117 or 118 are also supported (both work equally and are supported for compatibility reasons).If the value if the controller is lower than 64, the Engine Next is executed, otherwise EnginePrevious.

**EngineInsertAppendStep**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists, direct control of light channels and creation and editing of Cue Lists and Cue Steps. The command can be used to edit the data of a Cue Step.

The engines CueStepData, that can be modified with the **EngineSetCueStepDataType** command, sets the type of the Cue Step. If the Cue Step is a Scene (a Cue Scene for short) the engines light channels that are in edit mode is stored in the Cue Scene.

A Cue List can be created with the **CueListCreate** command. The engine should know which Cue List is the target. This can be done with the **EngineLoad** or the **EngineUsesCueList** commands.

The serial command number for **EngineInsertAppendStep** is 92 (5Ch hexadecimal). When appending a step at the end of a Cue List, the command has one 8-bit parameter, the engine number. When inserting a step somewhere in the Cue List the command has one extra parameter, the step number.

General form
* 5C ee [ss] #

| | |
|---|---|
| ee | Engine number (1…8) |
| ss | Step number (0…99) |

For example

* 5C 01 #
Appends a step to the current Cue List of engine A

* 5C 01 05 #
Inserts a step before step 5 in the current Cue List of engine A

**EngineInsertAppendStep**

The **EngineInsertAppendStep** command is implemented for MIDI as Controller message number 92. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is ignored. The Cue List number must be set prior to this with the 14 bit controller 21 (21 & 53).

**EngineReplaceStep**

The LanBox has multiple engines.Each engine can be used for sequencing Cue Lists, direct control of light channels and creation and editing of Cue Lists and Cue Steps. The command can be used to edit the data of a Cue Step.

The engines CueStepData, that can be modified with the **EngineSetCueStepDataType** command, sets the type of the Cue Step. If the Cue Step is a Scene (a Cue Scene for short) the engines light channels that are in edit mode is stored in the Cue Scene.

A Cue List can be created with the **CueListCreate** command. The engine should know which Cue List is the target. This can be done with the **EngineLoad** or the **EngineUsesCueList** commands.

The serial command number for **EngineReplaceStep** is 103 (67h hexadecimal). The command has two 8-bit parameters, the engine number and the step number that needs to be replaced.

General form
* 67 ee ss #

| | |
|---|---|
| ee | Engine number (1…8) |
| ss | Step number (0…99) |

For example
* 67 01 05 #
Replaces a step 5 in the current Cue List of engine A

**EngineReplaceStep**

The **EngineReplaceStep** command is implemented for MIDI as Controller message number 103. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is ignored. The Cue List number must be set prior to this with the 14 bit controller 21 (21 & 53).

**EngineSetDeviceID**

The LanBox has multiple engines.Each engine has its own unique device_ID number which is used in the MIDI Show Control (MSC) system and the SysEx File Dump protocol. When using more than one LanBox in a MSC system, each engine should have a unique device_ID. The EngineSetDeviceID command sets this ID. This ID is stored in non-volatile memory.

The serial command number for EngineSetDeviceID is 94 (5Eh hexadecimal). The command has two 8-bit parameters. The first parametsr is the engine number for which the ID must be set. The second is the ID. The ID should be 7 bit number.

General form
* 5E ee id #

| | |
|---|---|
| ee | Engine number (1…8) |
| id | Device ID (0…127) |

For example

* 5E 01 2C #
Sets the device ID of engine A to 44

**EngineSetDeviceID**

The EngineSetDeviceID command is implemented for MIDI as Controller message number 94. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the ID.

**EngineSetAutoOutput**

The LanBox has multiple engines.Each engine can make use of the auto output command. With auto output switched on (default) each change of value of a channel in that engine will automatically enable that channel.

The serial command number for EngineSetAutoOutput is 100 (64h hexadecimal). The command has two 8-bit parameters. The first parametsr is the engine number for which the auto output must be set. The second determine if the auto output will be turned Off (0), or On (>0).

General form
* 64 ee ae #

| | |
|---|---|
| ee | Engine number (1…8) |
| ae | 0= Off, >0 = On |

For example

* 64 01 FF #
Turns the auto output feature on in engine A

**EngineSetAutoOutput**

The EngineSetAutoOutput command is implemented for MIDI as Controller message number 100. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the ID.

**EngineSetCueListWait**

The LanBox has multiple engines.With each engine you can set or clear the wait status of a Cue Step. The **EngineSetCueListWait** sets or clears the wait status of the current Cue Step active in that engine.

The serial command number for **EngineSetCueListWait** is 97 (61h hexadecimal). The command has two 8-bit parameters. The first parametsr is the engine number for which the auto output must be set. The second determine if the wait status will be cleared (0), or set (>0).

General form
* 61 ee ae #

| | |
|---|---|
| ee | Engine number (1…8) |
| ae | 0= Clear, >0 = Set |

For example

* 61 01 FF #
Sets the wait status of the current Cue Step in engine A.

**EngineSetCueListWait**

The EngineSetCueListWait command is implemented for MIDI as Controller message number 97. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the ID.

**CueListCreate**

The LanBox has a file system that can store Cue Lists. Cue Lists form the heart of the LanBox automation principle and can contain not only lighting instructions but also execution instructions and mode instructions.

The **CueListCreate** command makes it possible to create a new empty Cue List in the file system. With commands like **CueListCreateStep** and **CueListCreateScene** the new Cue list can be filled with data.

The serial command number for **CueListCreate** is 95 (5Fh hexadecimal). The command has one 16-bit parameter, the Cue List number. If a Cue List with that number already exists, the command will fail. The old Cue List should first be removed with the **CueListRemove** command.

General form
* 5F cccc #
cccc Cue List number (1…500)

For example

* 5F 01 2C #
Creates a new Cue List with number 460

**CueListCreate**

The **CueListCreate** command is implemented for MIDI as Controller message number 95. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is ignored. The Cue List number must be set prior to this with the 14 bit controller 21 (21 & 53). If a Cue List with that number already exists, the command will fail. The old Cue List should first be removed with the **CueListRemove** command.

**CueListRemove**

The LanBox has a file system that can store Cue Lists. Cue Lists form the heart of the LanBox automation principle and can contain not only lighting instructions but also execution instructions and mode instructions.

The **CueListRemove** command makes it possible to remove an existing Cue List from the file system.

The serial command number for **CueListRemove** is 96 (60h hexadecimal). The command has one 16-bit parameter, the Cue List number. If a Cue List with that number does not exists, the command will fail.

General form
* 60 cccc #
cccc Cue List number (1…500)

For example

* 60 01 2C #
Removes the Cue List with number 460

**CueListRemove**

The **CueListRemove** command is implemented for MIDI as Controller message number 96. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is ignored. The Cue List number must be set prior to this with the 14 bit controller 21 (21 & 53). If a Cue List with that number does not exists, the command will fail.

**CueListRemoveStep**

The LanBox has a file system that can store Cue Lists. Cue Lists form the heart of the LanBox automation principle and can contain not only lighting instructions but also execution instructions and mode instructions.

The **CueListRemoveStep** command makes it possible to remove an existing step from an existing Cue List in the file system. When the step is remove all the steps after the removed step will move up one step to fill the hole. If the step was a Cue Scene, the Cue Scene file will automatically be removed along with the Step.

The serial command number for **CueListRemoveStep** is 98 (62h hexadecimal). The command has one 16-bit parameter, the Cue List number and a 8-bit parameter, the Cue Step number. If a Cue List with that number does not exists, the command will fail. If a Cue Step with that number does not exist, the command will fail.

General form
* 62 cccc ss #

| | |
|---|---|
| cccc | Cue List number (1…500) |
| ss | Cue Step number (1…99) |

For example
* 62 01 2C 33 #

Removes step 51 in the Cue List with number 460

**CueListRemoveStep**

The **CueListRemoveStep** command is implemented for MIDI as Controller message number 98. The MIDI channel on which the controller is send determines which engine's ID is set, the value of the controller is the step number. The Cue List number must be set prior to this with the 14 bit controller 21 (21 & 53). If a Cue List with that number does not exists, the command will fail. If a Cue Step with that number does not exist, the command will fail.

**CueListWrite**

The LanBox can store Cue Lists in its non-volatile memory. This memory is organized like a file system. With the **CueListWrite** command a Cue List can be stored, replaced or expanded in this file system.

Each Cue List can consist of up to 99 Cue Steps. When writing a Cue List you should pass on the total number of Cue Steps that will be written to that Cue List. Because the serial communication can handle only up to 500 characters (or 250 bytes) of data per frame and a Cue List can consist of up to 700 bytes of data, it may be required to write the Cue List in multiple frames. In the first frame, the 'Number of Cue Steps in Cue List' parameter should give the total number of steps. In the subsequent frames this parameter should be set to 0.

Each Cue List has a unique Cue List number. If you attempt to Write a Cue List that does already exist, the old Cue List will be replaced.

The serial command number for **CueListWrite** is 170 (AAh hexadecimal). The command has one 16-bit parameter, the Cue List number and a 8-bit parameter, the number of Cue Steps in a new list. If this number of steps parameter is 0, the steps will be added to an existing Cue List that has sufficient room.

General form
* AA cccc ss [s1 s2 s3 s4 s5 s6 s7] *
cccc Cue List number (1…500)
ss Number of steps in Cue List (0…99)

For example
* AA 01 C8 03 01 03 1F 2C 00 00 00 01 03 1B 1B 00 00 00 14 01 00 00 00 00 00 #

Will create a new Cue List 456 (01C8h) with a total of 3 Cue Steps
Cue Step 1 : show scene for 10s crossfading 3.0s
Cue Step 2 : show scene for 2.0s crossfading 2.0s
Cue Step 3 : Goto Cue Step 1

**CueListWrite**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CueListRead**

The LanBox can store Cue Lists in its non-volatile memory. This memory is organized like a file system. With the **CueListRead** command you get retrieve a Cue List from the file system.

The serial command number for **CueListRead** is 171 (ABh hexadecimal). The command has one 16-bit parameter, the Cue List number and a 8-bit parameter, the number of Cue Steps that need to be dumped. If this number is 0, all the Cue Steps in the Cue List will be dumped.

General form
* AB cccc ss ns #
    cccc  Cue List number (1…500)
    ss  Cue Step number to start with (0…99)
    ns  Number of steps to dump
* [s1 s2 s3 s4 s5 s6 s7] #

For example
* AB 01 C8 00 #

Will return all the Cue Steps of Cue List 456 (01C8h)
* 01 03 1F 2C 00 00 00  01 03 1B 1B 00 00 00  14 01 00 00 00 00 00 #

Cue Step 1 : show scene for 10s crossfading 3.0s
Cue Step 2 : show scene for 2.0s crossfading 2.0s
Cue Step 3 : Goto Cue Step 1

**CueListRead**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CueSceneWrite**

The LanBox can store Cue Lists and Cue Scenes in its non-volatile memory. This memory is organized like a file system. If a Cue List step is of the type Cue Scene (01), then it will have a Cue Scene file associated with it. You can write a new Cue Scene file into the file system with the **CueSceneWrite** command and also associate it with a Cue List and step.

A Cue Scene can consist of up to 250 channels. Because this would amount to a total size of over 500 bytes and the serial serial communication can handle only up to 500 characters (or 250 bytes) of data per frame, it may be needed to write the scene in more than one frame. In the first frame, the 'Number of channels in scene' should be set to the total number of channels that will be written to that scene. In each subsequent frame this parameter should be set to 0.

The Scene flags determine what type of scene this is. At this moment there are only two types defined:

0 : Absolute scene
1 : Relative scene

When the scene is an absolute scene, each channel value is an unsigned value. When the scene is an relative scene, each channel value is a signed, twos complement value.

General form
*AC cccc ss nn sf [cn cv] #

| | |
|---|---|
| cccc | Cue List number (1…500) |
| ss | Cue Step number (1…99) |
| sf | Scene flags |
| nn | Number of channels in scene |
| cn | Channel number |
| cv | Channel value |

* AC 01 C8 01 04 00 01 11 02 22 03 33 04 44 #
Will create a new absolute Cue Scene associated with Cue List 456 (01C8h) step 1
Channel 1 has a value of 17 (11h)
Channel 2 has a value of 34 (22h)
Channel 3 has a value of 51 (33h)
Channel 4 has a value of 68 (44h)

**CueSceneWrite**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CueSceneRead**

The LanBox can store Cue Lists & Cue Scenes in its non-volatile memory. This memory is organized like a file system. With the **CueSceneRead** command you get retrieve a Cue Scene from the file system.

The serial command number for **CueSceneRead** is 173 (ADh hexadecimal). The command has one 16-bit parameter, the Cue List number and a 8-bit parameter, the Cue Step number that need to be dumped.

The first byte is the Scene flag, the second byte is the number of channels. The rest of the bytes are channel number & channel value pairs.

General form:
* AD cccc ss #
    - cccc Cue List number (1…500)
    - ss Cue Step number (1…99)

* sf nn [cn cv] #
    - sf Scene flags
    - nn Number of channels in scene
    - cn Channel number
    - cv Channel value

For example
* AD 01 C8 01 #
Will return all the Cue Scene of Cue List 456 (01C8h) step 1

* 00 04 01 11 02 22 03 33 04 44 #
Scene is absolute (00)
Scene has 4 channels (04)
Channel 1 has a value of 17 (11h)
Channel 2 has a value of 34 (22h)
Channel 3 has a value of 51 (33h)
Channel 4 has a value of 68 (44h)

**CueSceneRead**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetMidi**

The LanBox can communicate through MIDI or serial. With this command the LanBox is forced to switch to one of then.

The serial command number for **CommonSetMidiMode** is 104 (68h hexadecimal). The command has one parameter, whether MIDI comm should be switched on or off.

General form
* 68 mm #
mm 00 MIDI mode Off, >0 MIDI mode On

For example
* 68 FF #
Switches the LanBox to MIDI communications

**CommonSetMidi**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetNumDmxChannels**

The LanBox can transmit up to 250 DMX channels. With this command the number of DMX channels that is transmitted can be set. Note that less DMX channel means better performance of the LanBox.

The serial command number for is 105 (69h hexadecimal). The command has one parameter, the number of DMX channels that should be transmitted.

General form
* 69 ch #
    ch  number of DMX channels (1…250)

For example
* 69 FA #
    Set the number of DMX channels to 250 (FAh)

**CommonSetNumDmxChannels**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetDmxOffset**

The LanBox can transmit up to 250 DMX channels. However it is possible to address DMX addresses above address 250. The LanBox can offset the DMX addresses with any amount between 0…255. For example if the DMX offset would be 100, DMX address 1 is shifted to DMX address 101. The first 100 DMX addresses will be send with a value of 0.

The serial command number for is 106 (6Ah hexadecimal). The command has one parameter, the DMX offset.

General form
* 6A ch #
ch DMX offset (0…255)

For example
* 6A 64 #
Set the DMX offset to 100 (64h)

**CommonSetDmxOffset**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetPatcher**

The LanBox works with light channels. Each light channel must eventually be mapped to a DMX channel. The patcher information is used for this translation. With the CommonGetPatcher a number of patch pairs can be returned.

The serial command number for **CommonGetPatcher** is 128 (80h hexadecimal). The command has two 8-bit parameters; first the DMX channel number for which the patcher data must be returned and the number of channels.

General form
* 80 ll cc #

| | |
|---|---|
| ll | first DMX channel number |
| cc | Number of DMX channels |

For example

* 80 05 02 #
Will return something like this:
* 66 82 #
DMX channel 5 is mapped to light channel 102 (66h)
DMX channel 6 is mapped to light channel 130 (82h)

**CommonGetPatcher**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetPatcher**

The LanBox works with light channels. Each light channel must eventually be mapped to a DMX channel. The patcher information is used for this translation.

The Patcher data consists of 250 values (one for each light channel). Each value indicates to which DMX channel the light channel must be mapped.

The serial command number for CommonSetPatcherTable is 129 (81h hexadecimal). The command can have any number of patch pair parameters. Each patch pair parameter consists of 2 bytes; DMX channel number and light channel number.

It is allowed to map one light channel to more than one DMX channel. The effect is that both DMX channels will always have the same values. It is also allowed to map a light channel to 0. The effect is that the light channel is not used. It is also allowed to map light channel 0 (which does not exist) to a DMX channel. The effect is that the DMX channel will always have the value 0.

General form
* 81 [dd ll] #

| | |
|---|---|
| dd | DMX channel number (1…250) |
| ll | Light channel patch number (0…250) |

For example
* 81 03 55 04 66 77 00 #
Maps DMX channel 3 on light channel 85 (55h)
Maps DMX channel 4 to light channel 102 (66h)
DMX channel 119 (77h) will always be 0
Light channel 5 is not used.

**CommonSetPatcher**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetGain**

Each DMX channel has a gain setting associated with it. With this gain the output level of the DMX channel can be defined. With the **CommonGetGain** command a number of gain pairs can be returned.

The serial command number for **CommonGetGain** is 130 (82h hexadecimal). The command has two 8-bit parameters; first the DMX channel number for which the gain data must be returned and the number of channels.

General form
* 82 dd cc #
    dd  first DMX channel number
    cc  Number of channels

For example

* 82 05 02 #
    Will return something like this:
* 82 82 #
DMX channel 5 has a gain of 102 (66h)
DMX channel 6 has a gain of 130 (82h)

**CommonGetGain**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetGain**

Each DMX channel has a gain setting associated with it. With this gain the output level of the DMX channel can be defined. With the CommonSetGain command the gain for each DMX channel can be defined.

The output level for a light channel is defined by the summation of the light channels in the 8 engines. Each engine supplies the value for any light channel (if the channel is active in that engine) to the mixer. The relation to the resulting mixer value and the DMX channel value is given by the Gain.

The gain value has a range of 0…255. The nominal (and default) value for the gain is 128. With this value the DMX channel value is equal to the (mixer) light channel value. Higher gain values will increase the DMX channel value, lower channel value will decrease it.

| Gain values | | DMX Output value |
|---|---|---|
| 0 | = | 0 |
| 64 | 0.5x | the light channel value |
| 128 | = | the light channel value |
| 192 | 1.5x | the light channel value |
| 255 | 2x | the light channel value |

The serial command number for CommonSetGain is 131 (83h hexadecimal). The command can have any number of gain pair parameters. Each gain pair parameter consists of 2 bytes; DMX channel number and gain.

General form
* 83 [dd gg] #

| | |
|---|---|
| dd | DMX channel number |
| gg | Gain value |

For example
* 83 05 82 06 44 #

Sets for DMX channel 5 a gain of 130 (82h)
Sets for DMX channel 6 a gain of 68 (44h)

**CommonSetGain**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetCurve**

Each DMX channel can have a curve associated with it. With a curve you can alter the mapping of light channels per value. Curves are often used to account for nonlinear behavior of lighting equipment. The LanBox can have 7 different, user defined, curves in addition to the default, build in 1-to-1 curve. With the CommonGetCurve command the assignment of a number of DMX channels can be returned.

The serial command number for CommonGetCurve is 132 (84h hexadecimal). The command has two 8-bit parameters; first the DMX channel number for which the curve data must be returned and the number of channels.

General form
* 84 dd cc #
    dd  first DMX channel number
    cc  Number of channels

For example

* 84 05 02 #
    Will return something like this:
* 01 00 #
DMX channel 5 is assigned to curve 1
DMX channel 6 is assigned to curve 0 (the default, build-in, 1-to-1 curve)

**CommonGetCurve**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetCurve**

Each DMX channel can have a curve associated with it. With a curve you can alter the mapping of light channels per value. Curves are often used to account for nonlinear behavior of lighting equipment. The LanBox can have 7 different, user defined, curves in addition to the default, build in 1-to-1 curve. With the CommonSetCurve command a curve can be assigned to a DMX channel.

The serial command number for CommonSetCurve is 133 (85h hexadecimal). The command can have any number of curve pair parameters. Each gain curve parameter consists of 2 bytes; DMX channel number and curve.

General form
* 85 [dd cc] #
dd DMX channel number
cc Curve number (0…7)

For example
* 85 05 01 06 00 #
Sets for DMX channel 5 curve number 1
Sets for DMX channel 6 curve number 0 (the default, build-in, 1-to-1 curve)

**CommonSetCurve**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetSlope**

Each DMX channel has a Slope limit assigned to it. The slope limit limits the amount of change per DMX frame for a DMX channel. The slope limit can be used for equipment that can not handle rapid changes in intensity. The slope limit is a number from 0…255. At the default value (255) the DMX channel will be allowed any amount of change in value. At a setting of 10 the value will only change with maximum of 10 (on a scale of 0…255) per DMX frame.

The serial command number for CommonGetSlope is 134 (86h hexadecimal). The command has two 8-bit parameters; first the DMX channel number for which the slope data must be returned and the number of channels.

General form
* 86 dd cc #
dd first DMX channel number
cc Number of channels

For example

* 86 05 02 #
Will return something like this:
* 10 ff #
DMX channel 5 has a slope limit of 16 (10h)
DMX channel 6 has a slope limit of 255 (ffh)

**CommonGetSlope**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetSlope**

Each DMX channel has a Slope limit assigned to it. The slope limit limits the amount of change per DMX frame for a DMX channel. The slope limit can be used for equipment that can not handle rapid changes in intensity. The slope limit is a number from 0…255. At the default value (255) the DMX channel will be allowed any amount of change in value. At a setting of 10 the value will only change with maximum of 10 (on a scale of 0…255) per DMX frame.

The serial command number for CommonSetSlope is 135 (87h hexadecimal). The command can have any number of slope pair parameters. Each gain slope parameter consists of 2 bytes; DMX channel number and slope.

General form
* 87 [dd ss] #
dd DMX channel number
ss slope number (0…255)

For example
* 87 05 10 06 ff #
Sets for DMX channel 5 a slope limit of 16 (10h)
Sets for DMX channel 6 a slope limit of 255 (ffh)

**CommonSetSlope**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetCurveTable (1…7)**

Each DMX channel can have a curve associated with it. With a curve you can alter the mapping of light channels per value. Curves are often used to account for nonlinear behavior of lighting equipment. The LanBox can have 7 different, user defined, curves in addition to the default, build in 1-to-1 curve. With the CommonGetCurveTable command you can get the currently defined curve.

The serial command number for CommonGetCurveTable is:

140 (8Ch hexadecimal) for table 1.
142 (8Eh hexadecimal) for table 2.
144 (90h hexadecimal) for table 3.
146 (92h hexadecimal) for table 4.
148 (94h hexadecimal) for table 5.
150 (96h hexadecimal) for table 6.
152 (98h hexadecimal) for table 7.

The command has two 8-bit parameters; first the DMX channel number for which the slope data must be returned and the number of channels.

General form
* 8C ff nn #
ff First input value (0…255)
nn Number of values (0…255) with 0 you'll get the complete curve!

For example
* 8C 00 04 #
Will return something like this:
* 00 02 04 06 08 #
Input intensity 0 is mapped to output intensity 0
Input intensity 1 is mapped to output intensity 2
Input intensity 2 is mapped to output intensity 4
Input intensity 3 is mapped to output intensity 6
Input intensity 4 is mapped to output intensity 8

**CommonGetCurveTable (1…7)**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetCurveTable (1…7)**

Each DMX channel can have a curve associated with it. With a curve you can alter the mapping of light channels per value. Curves are often used to account for nonlinear behavior of lighting equipment. The LanBox can have 7 different, user defined, curves in addition to the default, build in 1-to-1 curve. With the **CommonSetCurveTable** command a curve can be defined.

The serial command number for **CommonSetCurveTable** is:

141 (8Dh hexadecimal) for table 1.
143 (8Fh hexadecimal) for table 2.
145 (91h hexadecimal) for table 3.
147 (93h hexadecimal) for table 4.
149 (95h hexadecimal) for table 5.
151 (97h hexadecimal) for table 6.
153 (99h hexadecimal) for table 7.

The command can have any number of curve pair parameters. Each gain curve parameter consists of 2 bytes; input intensity and output intensity.

General form
* 8D [ii oo] #

| | |
|---|---|
| ii | Input intensity (0…255) |
| oo | Output intensity (0…255) |

For example
* 8D 00 00 01 02 02 04 03 06 04 08 #

Maps input intensity 0 to output intensity 0
Maps input intensity 1 to output intensity 2
Maps input intensity 2 to output intensity 4
Maps input intensity 3 to output intensity 6
Maps input intensity 4 to output intensity 8

**CommonSetCurveTable (1…7)**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonStorePostTable**

The Post table holds the data for the post mixer operations. The operations are; DMX Patch, Gain, Curve, Slope Limit. Each physical DMX channel has it's own set of parameters for the post mixer operations. The LanBox can store the patcher table in its file system. When a patcher file is found in the file system after boot, it is loaded.

The serial command number for **CommonStorePostTable** is 154 (9Ah hexadecimal). The command does not have any parameters.

General form
* 96 #

**CommonStorePostTable**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGet16BitTable**

The LanBox can make fades over 16 channels. In order to know which channels form 16 bit channels and what are the high and low channels, the LanBox maintains a 16 bit table. With the CommonSet16BitTable command 16 bit pair can be added and deleted from this list.

The serial command number for CommonGet16BitTable is 160 (A0h hexadecimal). The command does not have any parameters. It always returns a list of known 16 pairs, first the high channel and then the low channel.

General form
* A0 #

For example
* A0 #
Will return something like this:
* 10 11 22 25 #
Channel 16 (10h) and channel 17 (11h) form a 16 bit pair
Channel 34 (22h) and channel 37 (37h) form a 16 bit pair

**CommonGet16BitTable**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSet16BitTable**

The LanBox can make fades over 16 channels. In order to know which channels form 16 bit channels and what are the high and low channels, the LanBox maintains a 16 bit table. With the CommonSet16BitTable command 16 bit pair can be added and deleted from this list.

The serial command number for **CommonSet16BitTable** is 161 (A1h hexadecimal). The command can have any number of 16 pit pair sets. Each set is 3 bytes long. The first byte indicates whether the pair must be set or cleared, the other two identify the high and low channel of the pair.

If one of the elements of a 16 bit that is set with this command was already part of a 16 bit pair, the old pair is removed from the 16 bit table.

The two elements of the pair may not be too far apart from each other. A maximum of +7 or -8 channels must be observed.

General form
* A1 [mm hh ll] #
mm Mode (00 = Clear 16 bit pair, >0 = Set 16 bit pair)
hh High channel of 16 bit light channel pain
ll Low channel of 16 bit light channel pain

For example
* A1 FF 2C 33 FF 04 03 00 22 3A #
Makes a 16 bit pair of channels 44 and 51
Makes a 16 bit pair of channel 4 and 3
Removes the 16 bit pair of 34 and 58

**CommonSet16BitTable**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonStore16BitTable**

The LanBox can make fades over 16 channels. In order to know which channels form 16 bit channels and what are the high and low channels, the LanBox maintains a 16 bit table. The LanBox can store the 16 bit table in its file system. When a 16 bit file is found in the file system after boot, it is loaded in the 16 bit Table

The serial command number for CommonStore16BitTable is 162 (A2h hexadecimal). The command does not have any parameters.

General form
* A2 #

**CommonStore16BitTable**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetMIDIMapping**

In the LanBox each engine can be associated with one or two MIDI channels. The first MIDI channel is the primary, the second is the secondary. The secondary MIDI channel is only needed to manipulate light channels above 127.

The serial command number for CommonSetMIDIMapping is 163 (A3h hexadecimal). The command has no parameters. A list is returned containing all the assignments of the engines.

General form
* A3 #
Will return
* [pm sm]
pm Primary MIDI channel
sm Secondary MIDI channel

For example
* A3 #
* 02 03 00 00 00 00 00 00 01 09 00 00 00 00 00 00 # >
Engine A has MIDI channels 2 (primary) and 3 (secondary)
Engine B has no MIDI channels
Engine C has no MIDI channels
Engine D has no MIDI channels
Engine E has MIDI channels 1 (primary) and 9 (secondary)
Engine F has no MIDI channels
Engine G has no MIDI channels
Engine H has no MIDI channels

**CommonGetMIDIMapping**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetMIDIMapping**

In the LanBox each engine can be associated with one or two MIDI channels. The first MIDI channel is the primary, the second is the secondary. The secondary MIDI channel is only needed to manipulate light channels above 127.

The serial command number for CommonSetMIDIMapping is 164 (A4h hexadecimal). The command has three parameters; engine number, primary MIDI channel, secondary MIDI channel.

General form
* A4 [ee pm sm] #

| | |
|---|---|
| ee | Engine number |
| pm | Primary MIDI channel |
| sm | Secondary MIDI channel |

For example
* A4 01 02 03 #
Will assign MIDI channels 2 (primary) and 3 (secondary) to engine A.

**CommonSetMIDIMapping**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonStoreMIDIMapping**

In the LanBox each engine can be associated with one or two MIDI channels. The first MIDI channel is the primary, the second is the secondary. The secondary MIDI channel is only needed to manipulate light channels above 127. The LanBox can store the MIDI mapping in its file system. When a MIDI mapping file is found in the file system after boot, it is loaded.

The serial command number for CommonStoreMIDIMapping is 165 (A5h hexadecimal). The command does not have any parameters.

General form
* A5 #

**CommonStoreMIDIMapping**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetDirectory**

The LanBox stores all the Cue Lists in a file system. Just like normal disk based file systems you can ge a ‘directory’ of all stored Cue Lists. The CommonGetDirectory allows you to get this directory. Due to internal limitations you can get a maximum of 80 Cue Lists per command, therefore the commands lets you set a starting Cue List. It does not matter if this starting Cue List exists or not, if you execute the command starting with Cue List 100 and only 99 and 101 exist, the command will return Cue List 101.

The command will return a string with 3 bytes for each Cue List present in the file system. The first two give the Cue List number, the third the number of steps in the Cue List.

General form
* A7 cccc #

For example
* A7 01 23 #
Lists all Cue Lists starting with Cue List 291 (0123h). For example it may return:

* 01 33 05 01 34 56 01 40 04 *
Cue List 307 (0133h) with a length of 5 steps
Cue List 308 (0134h) with a length of 86 (56h) steps
Cue List 320 (0140h) with a length of 4 steps

**CommonGetDirectory**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**Common16BitMode**

While the classic LanBox-LC could only address up to 250 light channels, the new LCM and LCX boxes goes beyond that. This means that if you want to address light channels above 255, a channel should be a 16 bit value (High byte first). For LanBox-LCX and LCM there is a command **Common16BitMode** (65 hex ) in order to set the box so it will accept 16 bit channel addresses.

General form
* 65 aa #
aa 16 bit addresses (>0 = Enabled, 0 = Disabled)

For example
* 65 FF #

Turn On 16 bit mode for this stream.

**Common16BitMode**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonGetApplicationID**

Each LanBox product has an application ID and version number. The LanBox-LCII compatible serial command number for CommonGetApplicationID is 5 (05h hexadecimal). The command has no parameters.

General form
*00050000#
Will return
* aaaa vvvv #

| | |
|---|---|
| aaaa | Application ID |
| vvvv | Version number |

For example
* 00050000 #
Will return
* F8 FB 01 2C # >

F8FB is the application ID (-1797)
01 2C is the version number (300 or 3.00)

**Note:**
**F8F9 = LanBox-LCII**
**F8FB = LanBox-LC+**
**F8FD = LanBox-LCX**
**F8FF = LanBox-LCM**

**CommonGetApplicationID**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

**CommonSetBaudRate**

The LanBox-LCII compatible serial command number for CommonSetBaudRate is 6 (0006h hexadecimal). The command has one 8-bit parameter; the baud rate.

General form
* 0006 00bb #
bb New baud rate (0 = 38400, 1 = 19200, 2 = 9600)

For Example
* 00 06 00 01 #
Sets the baud rate to 19200

**CommonSetBaudRate**

In MIDI this command is implemented only as SysEx command, see MIDI SysEx Commands.

# LanBox

# MIDI Show Control

# Commands

**Overview**

| | |
|---|---|
| • GO | 01 |
| • STOP | 02 |
| • RESUME | 03 |
| • TIMED_GO | 04 |
| • LOAD | 05 |
| • SET | 06 |
| • FIRE | 07 |
| • ALL_OFF | 08 |
| • RESTORE | 09 |
| • RESET | 0A |
| • GO_OFF | 0B |
| • GO_JAM | 10 |
| • STANDBY_+ | 11 |
| • STANDBY_- | 12 |
| • SEQUENCE_+ | 13 |
| • SEQUENCE_- | 14 |

**MIDI Show Control GO**

The GO command starts a Cue List at a Cue Step in an engine. The engine is determined by the device_ID. the Cue List number and the Cue Step number are determined by the Q_Number. Q_list and Q_path is ignored by the LanBox.

Both Cue List number and Cue Step number are send as ASCII numbers, separated by an ASCII decimal point (2E). For example Cue List 23, Cue Step 5 would be:

32 33 2E 35.

When a valid Cue List number and Cue Step number are received, the engine will set the Output, Fade and Sequencing attributes on and start executing the Cue List.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 01 | GO command |
| <Q_number> | |
| 00 | Delimiter |
| <Q_list> | |
| 00 | Delimiter |
| <Q_path> | |
| F7 | End of SysEx |

For example, Go Cue List 23 Cue Step 5 in engine with ID 4

F0 7F 04 02 7F 01 32 33 2E 35 F7

**MIDI Show Control STOP**

The STOP command halt execution of a Cue List in an engine. The engine is determined by the device_ID. Since there can be only one Cue List active in an engine, the LanBox will ignore the Q_number, Q_list and Q_path parameters. Execution can be resumed with the RESUME command.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 02 | STOP command |
| <Q_number> | |
| 00 | Delimiter |
| <Q_list> | |
| 00 | Delimiter |
| <Q_path> | |
| F7 | End of SysEx |

For example, Stop engine with ID 4

F0 7F 04 02 7F 02 F7

**MIDI Show Control RESUME**

The RESUME command resumes the execution of a Cue List in an engine that was stopped by the STOP command. The engine is determined by the device_ID. Since there can be only one Cue List active in an engine, the LanBox will ignore the Q_number, Q_list and Q_path parameters. Execution can be resumed with the RESUME command.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 03 | RESUME command |
| <Q_number> | |
| 00 | Delimiter |
| <Q_list> | |
| 00 | Delimiter |
| <Q_path> | |
| F7 | End of SysEx |

For example, Resume engine with ID 4

F0 7F 04 02 7F 03 F7

### MIDI Show Control TIMED_GO

The TIMED_GO command starts a Cue List at a Cue Step in an engine at a certain time. The engine is determined by the device_ID. the Cue List number and the Cue Step number are determined by the Q_Number. Q_list and Q_path is ignored by the LanBox.

As long as the LanBox does not have an internal timer, the time data is ignored. For more info on the Standard Time Code format please refer to MIDI Show Control 1.1 specs page 6.

Both Cue List number and Cue Step number are send as ASCII numbers, separated by an ASCII decimal point (2E). For example Cue List 23, Cue Step 5 would be:

32 33 2E 35.

When a valid Cue List number and Cue Step number are received, the engine will set the Output, Fade and Sequencing attributes on and start executing the Cue List.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 04 | TIMED_GO command |
| hr mn sc fr ff | Standard Time Specification |
| <Q_number> | |
| 00 | Delimiter |
| <Q_list> | |
| 00 | Delimiter |
| <Q_path> | |
| F7 | End of SysEx |

For example, Go Cue List 23 Cue Step 5 at 20:39.40.5 in engine with ID 4

F0 7F 04 02 7F 04 14 27 28 05 00 32 33 2E 35 F7

### MIDI Show Control LOAD

The LOAD command load a Cue List at a Cue Step in an engine. The engine is determined by the device_ID. the Cue List number and the Cue Step number are determined by the Q_Number. Q_list and Q_path is ignored by the LanBox.

Both Cue List number and Cue Step number are send as ASCII numbers, separated by an ASCII decimal point (2E). For example Cue List 23, Cue Step 5 would be:

32 33 2E 35.

When a valid Cue List number and Cue Step number are received, the engine will NOT set the Output, Fade and Sequencing attributes on and will wait for a GO command.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 05 | LOAD command |
| <Q_number> | |
| 00 | Delimiter |
| <Q_list> | |
| 00 | Delimiter |
| <Q_path> | |
| F7 | End of SysEx |

For example, Go Cue List 23 Cue Step 5 in engine with ID 4

F0 7F 04 02 7F 05 32 33 2E 35 F7

### MIDI Show Control SET

The SET command defines a value of a generic control. For the LanBox implementation all the functions that can be set with standard MIDI controllers, also work with the SET command. The controller number mapped to the LSB of the Generic Control Number and the controller value mapped to the LSB of the Generic Control Value.

As long as the LanBox does not have an internal timer, the time data is ignored. For more info on the Standard Time Code format please refer to MIDI Show Control 1.1 specs page 6.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 06 | SET command |
| cc cc | Generic Ctrl Number, LSB first |
| vv vv | Generic Ctrl Value, LSB first |
| hr mn sc fr ff | Standard Time Spec, optional |
| F7 | End of SysEx |

For example, Set Chase Mode (75, 4Bh) to Chase Up Repeat (17, 11h) in engine with ID 4
F0 7F 04 02 7F 06 4B 00 11 00 F7

### MIDI Show Control FIRE

The FIRE command triggers a preprogrammed macro in the LanBox. The first 128 Cue List can be used as macros. Each macro can start Cue List in various engines, or set engines in certain modes. Macros can be programmed the same way you would program a Cue List.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 07 | FIRE command |
| mm | Macro number |

For example, Fire Macro 33 (21h) in engine with ID 4
F0 7F 04 02 7F 07 21 F7

### MIDI Show Control ALL_OFF

The ALL_OFF command turns off all status bits (Output, Sequencing, Fading, Soloing) of the engine, but remembers these setting. Settings can be restored with the RESTORE command.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 08 | ALL_OFF command |

For example, ALL_OFF in engine with ID 4
F0 7F 04 02 7F 08 F7

### MIDI Show Control RESTORE

The RESTORE command restores all status bits (Output, Sequencing, Fading, Soloing) of the engine to the state prior to the ALL_OFF.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 09 | RESTORE command |

For example, RESTORE in engine with ID 4
F0 7F 04 02 7F 09 F7

### MIDI Show Control RESET

The RESET command terminates all running cues and resets the engine to the initialized state equivalent to a newly powered-up condition.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 0A | RESTORE command |

For example, RESTORE in engine with ID 4
F0 7F 04 02 7F 0A F7

### MIDI Show Control GO_OFF

The GO_OFF command does the same thing as the GO command, starts a Cue List at a Cue Step in an engine. The engine is determined by the device_ID. the Cue List number and the Cue Step number are determined by the Q_Number. Q_list and Q_path is ignored by the LanBox.

In MSC the GO_OFF command can be used for devices that can't automatically replace a running Cue List with a new one. Since the LanBox can do this anyway, the GO_OFF command works the same way as the GO command.

Both Cue List number and Cue Step number are send as ASCII numbers, separated by an ASCII decimal point (2E). For example Cue List 23, Cue Step 5 would be:

32 33 2E 35.

When a valid Cue List number and Cue Step number are received, the engine will set the Output, Fade and Sequencing attributes on and start executing the Cue List.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 0B | GO_OFF command |
| <Q_number> | |
| 00 | Delimiter |
| <Q_list> | |
| 00 | Delimiter |
| <Q_path> | |
| F7 | End of SysEx |

For example, Go Cue List 23 Cue Step 5 in engine with ID 4

F0 7F 04 02 7F 0B 32 33 2E 35 F7

### MIDI Show Control GO_JAM

The GO_OFF command does the same thing as the GO command, starts a Cue List at a Cue Step in an engine but also forces the clock time to the Go Time of that Cue List. The engine is determined by the device_ID. the Cue List number and the Cue Step number are determined by the Q_Number. Q_list and Q_path is ignored by the LanBox.

As long as the LanBox does not have an internal timer, this command will do exactly the same as the GO command.

Both Cue List number and Cue Step number are send as ASCII numbers, separated by an ASCII decimal point (2E). For example Cue List 23, Cue Step 5 would be:

32 33 2E 35.

When a valid Cue List number and Cue Step number are received, the engine will set the Output, Fade and Sequencing attributes on and start executing the Cue List.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 10 | GO_OFF command |
| <Q_number> | |
| 00 | Delimiter |
| <Q_list> | |
| 00 | Delimiter |
| <Q_path> | |
| F7 | End of SysEx |

For example, Go Cue List 23 Cue Step 5 in engine with ID 4

F0 7F 04 02 7F 10 32 33 2E 35 F7

### MIDI Show Control STANDBY_+

The STANDBY_+ command places into standby position the next Cue Step in the current Cue List. The current Cue List keeps running (if it was running), but when the duration time is over the Cue Step placed in standby is executed.

If the optional Q_List parameter is specified, this parameters holds the Cue Step that needs to be placed in standby.

The Cue Step number is send as an ASCII number. For example Cue Step 5 would be:

35.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 11 | STANDBY_+ command |
| <Q_list> | |
| F7 | End of SysEx |

For example, STANDBY_+ in engine with ID 4
F0 7F 04 02 7F 11 F7

For example, STANDBY_+ Cue Step 5 in engine with ID 4
F0 7F 04 02 7F 11 35 F7

**MIDI Show Control STANDBY_-**

The STANDBY_- command places into standby position the previous Cue Step in the current Cue List. The current Cue List keeps running (if it was running), but when the duration time is over the Cue Step placed in standby is executed.

If the optional Q_List parameter is specified, this parameters holds the Cue Step that needs to be placed in standby.

The Cue Step number is send as an ASCII number. For example Cue Step 5 would be:

35.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 12 | STANDBY_+ command |
| <Q_list> | |
| F7 | End of SysEx |

For example, STANDBY_- in engine with ID 4
F0 7F 04 02 7F 12 F7

For example, STANDBY_- Cue Step 5 in engine with ID 4
F0 7F 04 02 7F 12 35 F7

**MIDI Show Control SEQUENCE_+**

The SEQUENCE_+ command places into standby position the next available Cue List. The current Cue List keeps running (if it was running), but when the duration time for the current Cue Step is over the Cue List placed in standby is executed.

If the optional Q_List parameter is specified, this parameters holds the Cue List that needs to be placed in standby.

Both Cue List number and Cue Step number are send as ASCII numbers, separated by an ASCII decimal point (2E). For example Cue List 23, Cue Step 5 would be:

32 33 2E 35.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 13 | STANDBY_+ command |
| <Q_list> | |
| F7 | End of SysEx |

For example, SEQUENCE_+ in engine with ID 4
F0 7F 04 02 7F 13 F7

For example, SEQUENCE_+ Cue List 23.5 in engine with ID 4
F0 7F 04 02 7F 13 32 33 2E 35 00 00 F7

**MIDI Show Control SEQUENCE_-**

The SEQUENCE_- command places into standby position the previous available Cue List. The current Cue List keeps running (if it was running), but when the duration time for the current Cue Step is over the Cue List placed in standby is executed.

If the optional Q_List parameter is specified, this parameters holds the Cue List that needs to be placed in standby.

Both Cue List number and Cue Step number are send as ASCII numbers, separated by an ASCII decimal point (2E). For example Cue List 23, Cue Step 5 would be:

32 33 2E 35.

General form:

| | |
|---|---|
| F0 7F <device_ID> 02 7F | General MSC header |
| 14 | STANDBY_+ command |
| <Q_list> | |
| F7 | End of SysEx |

For example, SEQUENCE_+ in engine with ID 4

F0 7F 04 02 7F 14 F7

For example, SEQUENCE_+ Cue List 23.5 in engine with ID 4

F0 7F 04 02 7F 14 32 33 2E 35 00 00 F7

# LanBox

# UDP network

# Commands

## UDP network Commands

From firmware version 2.xx for the LanBox-LCX and LCE, a UDP protocol is supported in order to broadcast DMX buffers, or to set them directly from third party applications like MAX. For network and UDP setup see the LCedit+ tutorial and the LanBox user manual. The default UDP port used is 4777, and all integers are in "network order" (most significant byte first).

A packet consists of a 4-byte packet header followed by one or more messages. If a message has odd length, then a padding byte is added to make sure the next message is at a 16-bit aligned offset. This padding may be omitted after the last message of course.

**Packet header C0 B7 sq sq**
16-bit cookie
16-bit sequence number

The cookie must always be 0xC0B7. Packets with a different cookie should be silently discarded. The sequence number is incremented by 1 for each packet, and may optionally be used to discard packets that are received out of sequence. (The LanBox does not currently do this though)

Two different types of messages are currently defined:

**Buffer broadcast C9 id ln ln ch ch <data>**
8-bit message type, always 0xC9
8-bit source buffer id
16-bit message length (of entire message, except padding)
16-bit channel offset
n bytes of data, where n = message length - 6

This message announces the contents of (part of) one of the buffers of the transmitting LanBox. The channel offset is the channel number corresponding to the first byte of data.

A LanBox can be configured to transmit the following buffers:
252 DMX Input Buffer
253 Analog/Switch Inputs
254 Mixer Buffer
255 DMX Output Buffer

A LanBox can be configured to copy incoming mixer buffer broadcasts of some other LanBox into its own mixer buffer or one of its layers.

**Buffer write CA id ln ln ch ch <data>**
8-bit message type, always 0xCA
8-bit target buffer id, see below for a list of ids
16-bit message length (of entire message, except padding)
16-bit channel offset
n bytes of data, where n = message length - 6

This message is almost the same, except in opposite direction: it is sent to a LanBox to directly write into one of its buffers. The buffer id must be either 254 to write into the mixer buffer, or the id of a layer to write into, in range 1 (layer A) through 63 (layer BK).

It possible that future versions of the protocol will define more types of messages. These will however all start with the general form:
8-bit message type
8-bit value
16-bit message length

If the message type of a message is neither 0xC9 nor 0xCA, then the message length can be used to skip over the message. If a message is encountered with length set to 0, parsing must abort to avoid an infinite loop.

# LanBox

# Cuestep

# Commands

### General

CueLists hold the automation data for the engine. A CueList can be executed by any of the 8 engines (and even by more than one at a time) and can contain lighting level information (CueScenes), sequencing information (start, stop, goto, loop, halt, resume, wait, etc) and engine status information (Output, Fade, Solo, Sequencing attributes, Mix modes, Chase mode and speed, Fade mode and speed, etc).

CueLists are use to perform functions automatically that could also be done manually. One CueList is special: CueList 1. When the LanBox starts-up, it looks for CueList 1. If it is found it will start to execute this CueList in Engine A. This makes stand alone operation possible.

CueLists are generally used for storing often used or very fast lighting movements or patterns. Because CueLists can contain timing information and synchronisation information, it is possible to use them together with outside controlling devices (such as a show control program).

Each CueList consists of up to 100 CueSteps. Each CueStep can contain one command or one CueScene. Each CueStep consists of a CueStep type and up to 6 parameters. In the file structure, each CueStep takes up 7 bytes.

| CueStep Types: | | Parameters: |
|---|---|---|
| 1) | CueScene | Fade Type, Fade Time, Hold Time |
| 2) | CueRefrenceScene | Fade Type, Fade Time, Hold Time, CueList, CueStep |
| 10) | GoEngine | EngineNum, CueList, CueStep |
| 11) | ClearEngine | EngineNum |
| 12) | SuspendEngine | EngineNum |
| 13) | ResumeEngine | EngineNum |
| 14) | StartEngine | EngineNum |
| 15) | StopEngine | EngineNum |
| 20) | GotoCueStep | CueStep |
| 21) | GoNext | EngineNum |
| 22) | GoPrevious | EngineNum |
| 23) | LoopTo | CueStep, NumRepeat |
| 24) | WaitEngine | HoldTime |
| 25) | WaitSMPTE | SMPTE time |
| 26) | WaitPulse | Polarity |
| 30) | SetEngineAttributes | Fade, Output, Solo |
| 31) | SetEngineMixMode | MixMode, Transparency, Fade Time |
| 32) | SetEngineChase | ChaseMode, ChaseSpeed, Fade Time |

**1) CueScene**
This command transfers lighting information of a scene file to the engine in which the CueLists is executed. Enables all channels used in the scene file. If a fade type and fade time is defined, for each lighting channel in the scene file a fader will be assigned. The engine will wait the amound of time defined by Hold Time before stepping to the next CueStep in the CueList.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | Fade type | (0…7) (Off, In, Out, X, Off, In CR, Out CR, X CR) |
| 2 | Fade time | (0…91) |
| 3 | Hold time | (0…91) |
| 4 | Scenefile bankNum* | |
| 5 | Scenefile Addr. high* | |
| 6 | Scenefile Addr. low* | |

*These parameters are only used internally by the file system and should only be changed by the file system. They determin the location of the Scene file in the filesystem.

**2) CueRefrenceScene**
This command transfers lighting information of a scene file defined in another CueList to the engine in which the CueLists is executed. Enables all channels used in the scene file. If a fade type and fade time is defined, for each lighting channel in the scene file a fader will be assigned. The engine will wait the amound of time defined by Hold Time before stepping to the next CueStep in the CueList.

CueReference scenes can be used to obtain global reference information from a scene. For example, the x-y position of a (number of) scan(s) for a certain object on the stage can be stored as a CueStep in a CueList 2 on CueStep 4. When you need the scans to go to that position while executing another CueList (say 10) use a CueReferenceScene 2.4 command. You now only have to adjust CueList 2 CueStep 4 and everywhere that position is used, it will change occordingly. This also goes for colors, gobos, intensities, whatever...

CueStepData bytes:

| | | |
|---|---|---|
| 1 | Fade type | (0…7) (Off, In, Out, X, Off, In CR, Out CR, X CR) |
| 2 | Fade time | (0…91) |
| 3 | Hold time | (0…91) |
| 4 | CueListNum high | |
| 5 | CueListNum low | |
| 6 | CueStep | (1…99) |

**10) GoEngineCueList**
This command loads a CueList at a CueStep in an engine. If the engine was running a CueList,it will continue running with the destinated cuelist and step  If the engine number is 0, then the engine executing this CueStep is the target.

This command will not set any attribute in the engine. If the status of these attributes is not known, use the **SetEngineAttributes** command to set them.
This command will not set the MixMode of the engine. If the status of te mixmode is not known, use the **SetEngineMixMode** command to set it.
This command will not set the Chase mode of the engine. If the status of the Chase mode is not known, use the **SetEngineChase** command to set it

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (0…8) |
| 2 | CueListNum high | |
| 3 | CueListNum low | |
| 4 | CueStep | (1…99) |
| 5 | - | |
| 6 | - | |

**11) ClearEngine**
This command clears all data, enables and outputs in an engine. If the engine number is 0, then the engine executing this CueStep is the target.

This command will clear all the enable bits of the engines channels.
This command will set all the data of the engines channels to zero.
This command will not change any other parameters of the engine.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (0…8) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**12) SuspendEngine**
This command suspends execution of a CueList at a CueStep in an engine. Faders currently running will stop running, hold time will stop counting down, the entire engine is frozen. The engine will remain this way until a ResumeEngine (or a StartEngine, StopEngine) command is executed by another engine. If the engine number is 0, then the engine executing this CueStep is the target.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (0…8) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**13) ResumeEngine**
This command resumes execution of a CueList at a CueStep in an engine that was previously suspended by SuspendEngine. Fades will start running again and hold time will commence counting down.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (1…8) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**14) StartEngine**
This command turns on the sequencer  in an engine.

This command will set the Sequencing attribute in the engine.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (1…8) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**15) StopEngine**
This command turns off the sequencer in an engine.

This command will clear the Sequencing attribute in the engine.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (1…8) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**20) GotoCueStep**
This command causes the engine to continue execution of the current CueList at the defined CueStep.

This command will not change any parameters of the engine (except the current CueStep).

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (1…8) |
| 2 | CueStep | (1…99) |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**21) GoNext**
This command will cause the target engine to continue execution of the current CueList in that engine at the next CueStep.

This command will not change any parameters of the engine (except the current CueStep).

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (1…8) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**22) GoPrevious**
This command will cause the target engine to continue execution of the current CueList in that engine at the previous CueStep.

This command will not change any parameters of the engine (except the current CueStep).

CueStepData bytes:

| | | |
|---|---|---|
| 1 | EngineNum | (1…8) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**23) LoopTo**
This command will cause the target engine to jump execution of the current CueList in that engine at the defined CueStep for as many times as defined in NumRepeat. After all the repeats are done the engine will continue execution of the CueList at the next CueStep.

This command will not change any parameters of the engine (except the current CueStep).

CueStepData bytes:

| | | |
|---|---|---|
| 1 | CueStep | (1…99) |
| 2 | NumRepeat | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**24) WaitEngine**
This command will cause the engine to wait an amount of time defined by HoldTime before continuing with the next CueStep in the current CueList.

This command will not change any parameters of the engine.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | HoldTime | (0…91) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**25) WaitSMPTE * NOT IMPLEMENTED YET***
This command will cause the engine to wait until the internal SMPTE timer has reached (or passed) a given time before continuing with the next CueStep in the current CueList.. The internal SMPTE timer is slaved to an external source.

This command will not change any parameters of the engine.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | SMPTE hours | (0…32) |
| 2 | SMPTE minutes | (0…59) |
| 3 | SMPTE seconds | (0…59) |
| 4 | SMPTE frames | (0…120) |
| 5 | - | |
| 6 | - | |

**26) WaitSwitch**
This command will cuase the engine to wait until a positive or negative level is found at pins 2 & 3 of the serial connector before continuing with the next CueStep in the current CueList.

This command will not change any parameters of the engine.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | Polarity | (0…63 = negative, 64…255 = positive) |
| 2 | - | |
| 3 | - | |
| 4 | - | |
| 5 | - | |
| 6 | - | |

**30) SetEngineAttributes**
This command sets the attributes af an engine. If the engine number is 0, then the engine executing this CueStep is the target.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | Engine | (0…8) |
| 2 | Fade | (0…63 = Off, 64…255 = On) |
| 3 | Output | (0…63 = Off, 64…255 = On) |
| 4 | Solo | (0…63 = Off, 64…255 = On) |
| 5 | - | |
| 6 | - | |

**31) SetEngineMixMode**
This command sets the mixmode of an engine. If the Mix mode is Transparent, then the Transparency byte and the Fade Time are important (in other cases they are ignored). If the fade time is not 0 and the mix mode is Transparent, the transparency depth will fade from transparency start to transparency end in the Fade time. If no fade type was defined, transparency end will be used. If the engine number is 0, then the engine executing this CueStep is the target.

If the engine number is 0, then the engine executing this CueStep is the target.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | Engine | (0…8) |
| 2 | MixMode | (0=Copy, 1=Mix-Up, 2=Mix-Down, 3=Transparent) |
| 3 | Transparency start | (0…255) |
| 4 | Transparency end | (0…255) |
| 5 | Fade Time | (0…91) |
| 6 | - | |

**32) SetEngineChase**
This command sets the chase parameters of an engine. If the fade time is not 0 the chase speed will fade from ChaseSpeed start to ChaseSpeed end in the Fade time. If no fade type was defined, transparency end will be used. If the engine number is 0, then the engine executing this CueStep is the target.

CueStepData bytes:

| | | |
|---|---|---|
| 1 | Engine | (0…8) |
| 2 | ChaseMode | (0, 1, 2, 3, 4, 17, 18, 19, 20) |
| 3 | ChaseSpeed start | (0…255) |
| 4 | ChaseSpeed end | (0…255) |
| 5 | Fade Time | (0…91) |
| 6 | - | |

**LanBox-LC + Time Encoding Table**

| Code | Sec | Code | Sec | Code | Sec | Code | Min | Code | Min |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.05 | 20 | 1.0 | 44 | 10 | 63 | 1.0 | 87 | 10 |
| 2 | 0.10 | 21 | 1.1 | 45 | 11 | 64 | 1.1 | 88 | 11 |
| 3 | 0.15 | 22 | 1.2 | 46 | 12 | 65 | 1.2 | 89 | 12 |
| 4 | 0.20 | 23 | 1.3 | 47 | 13 | 66 | 1.3 | 90 | 13 |
| 5 | 0.25 | 24 | 1.5 | 48 | 15 | 67 | 1.5 | 91 | 15 |
| 6 | 0.30 | 25 | 1.6 | 49 | 16 | 68 | 1.6 | | |
| 7 | 0.35 | 26 | 1.8 | 50 | 18 | 69 | 1.8 | | |
| 8 | 0.40 | 27 | 2.0 | 51 | 20 | 70 | 2.0 | | |
| 9 | 0.45 | 28 | 2.2 | 52 | 22 | 71 | 2.2 | | |
| 10 | 0.50 | 29 | 2.4 | 53 | 24 | 72 | 2.4 | | |
| 11 | 0.55 | 30 | 2.7 | 54 | 27 | 73 | 2.7 | | |
| 12 | 0.60 | 31 | 3.0 | 55 | 30 | 74 | 3.0 | | |
| 13 | 0.65 | 32 | 3.3 | 56 | 33 | 75 | 3.3 | | |
| 14 | 0.70 | 33 | 3.6 | 57 | 36 | 76 | 3.6 | | |
| 15 | 0.75 | 34 | 3.9 | 58 | 39 | 77 | 3.9 | | |
| 16 | 0.80 | 35 | 4.3 | 59 | 43 | 78 | 4.3 | | |
| 17 | 0.85 | 36 | 4.7 | 60 | 47 | 79 | 4.7 | | |
| 18 | 0.90 | 37 | 5.1 | 61 | 51 | 80 | 5.1 | | |
| 19 | 0.95 | 38 | 5.6 | 62 | 56 | 81 | 5.6 | | |
| | | 39 | 6.2 | 63 | 60 | 82 | 6.2 | | |
| | | 40 | 6.8 | | | 83 | 6.8 | | |
| | | 41 | 7.5 | | | 84 | 7.5 | | |
| | | 42 | 8.2 | | | 85 | 8.2 | | |

**LanBox-LCM pinning of RJ45 connector and adapters.**

| RJ45 (Female) | LCM Signal | Input/Output (On LCM) | DB9 (Female) | DTE Signal | Wiring Color1 | Wiring Color2 | DB25 (Male) | DCE Signal |
|---|---|---|---|---|---|---|---|---|
| 1 | SW0 | Input | 4 | DTR | Black | Blue | 6 | DSR |
| 2 | SW1 | Input | | | Yellow | Orange | 8 | CD |
| 3 | -MIDI | Output | 6 | DSR | Orange | Black | 20 | DTR |
| 4 | Gnd | - | 5 | Gnd | Red | Red | 7 | Gnd |
| 5 | Rx | Input | 3 | TxD | Green | Green | 3 | RxD |
| 6 | +MIDI, Tx | Output | 2 | RxD | Brown | Yellow | 2 | TxD |
| 7 | -MIDI | Input (opto) | 7 | RTS | Grey | Brown | 5 | CTS |
| 8 | +MIDI | Input (opto) | 8 | CTS | Blue | White | 4 | RTS |
| | | | | | | | | |
| | | | | | | | | |